

UNIVERSITÉ DU QUÉBEC À MONTRÉAL

PASIGRAPH : UN ENSEMBLE D'ALGORITHMES DE SOUS-ISOMORPHISME MAXIMAL EN MÉMOIRE
DISTRIBUÉE MET EN ÉVIDENCE LES VARIATIONS D'EMPILEMENT DANS LES MODULES STRUCTURAUX
GÉNÉRAUX D'ARN.

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUES

PAR
KOSSI WILFRIED AGBETO

AOÛT 2024

UNIVERSITÉ DU QUÉBEC À MONTRÉAL
Service des bibliothèques

Avertissement

La diffusion de ce mémoire se fait dans le respect des droits de son auteur, qui a signé le formulaire *Autorisation de reproduire et de diffuser un travail de recherche de cycles supérieurs* (SDU-522 – Rév.12-2023). Cette autorisation stipule que «conformément à l'article 11 du Règlement no 8 des études de cycles supérieurs, [l'auteur] concède à l'Université du Québec à Montréal une licence non exclusive d'utilisation et de publication de la totalité ou d'une partie importante de [son] travail de recherche pour des fins pédagogiques et non commerciales. Plus précisément, [l'auteur] autorise l'Université du Québec à Montréal à reproduire, diffuser, prêter, distribuer ou vendre des copies de [son] travail de recherche à des fins non commerciales sur quelque support que ce soit, y compris l'Internet. Cette licence et cette autorisation n'entraînent pas une renonciation de [la] part [de l'auteur] à [ses] droits moraux ni à [ses] droits de propriété intellectuelle. Sauf entente contraire, [l'auteur] conserve la liberté de diffuser et de commercialiser ou non ce travail dont [il] possède un exemplaire.»

REMERCIEMENTS

Je tiens à exprimer ma sincère gratitude envers toutes les personnes qui ont contribué à la réalisation de ce mémoire.

Tout d'abord, je voudrais remercier Reinharz Vladimir et Coti Camille pour leur guidance précieuse, leurs conseils éclairés et leur soutien constant tout au long de ce projet.

Je suis également reconnaissant envers ma famille pour leur amour, leur soutien indéfectible et leur compréhension tout au long de ce voyage académique.

Je suis profondément reconnaissant envers chacune de ces personnes qui ont contribué de près ou de loin à la réalisation de ce mémoire. Leur soutien et leur encouragement ont été un moteur essentiel dans la concrétisation de ce travail.

TABLE DES MATIÈRES

TABLE DES FIGURES	v
LISTE DES TABLEAUX	vii
RÉSUMÉ	viii
INTRODUCTION	1
0.1 Motifs structuraux d'ARN 3D.....	1
0.2 Problème d'extraction des motifs structuraux	2
0.3 Contribution	3
0.4 Structure du document	4
CHAPITRE 1 ARN	5
1.1 Modélisation informatique	5
1.1.1 Les Graphes	5
1.1.2 Le problème de trouver tous les MCCPS entre 2 graphes	7
1.2 Structure de l'ARN	10
1.2.1 L'ARN comme graphe	15
1.2.2 RIN (Réseaux d'interaction récurrents)	16
CHAPITRE 2 MÉTHODOLOGIE	18
2.1 Algorithme pour les Sous-graphes Partiels Communs Connexes Maximaux (MCCPS)	18
2.1.1 Recherche de toutes les correspondances d'arêtes.....	18
2.1.2 Extension des correspondances d'arêtes	19
2.2 Parallélisation de l'algorithme pour les Sous-graphes Partiels Communs Connexes Maximaux (MCCPS).....	26
2.2.1 Algorithme sur mémoire partagée	26
2.2.2 Algorithme sur mémoire distribuée	37

2.3	Des MCCPS aux RINs	37
2.4	Recherche des MCCPS contenant un motif donné	37
CHAPITRE 3 RÉSULTATS		39
3.1	Performance de la parallélisation	40
3.2	Description des RINs trouvés	44
3.2.1	RINs Locaux	44
3.2.2	RINs Globaux	46
3.3	Comparaison avec les travaux précédents	50
3.3.1	Anciens Rins locaux.....	50
3.3.2	Anciens Rins globaux	50
CONCLUSION.....		51
ANNEXE A ANNEXE		53
BIBLIOGRAPHIE		55

TABLE DES FIGURES

Figure 0.1	Structure tertiaire d'un ARN et sa représentation sous forme de graphe. Les interactions en bleu sont locales à un élément de la structure secondaire, et les interactions en rouge sont à longue portée.	2
Figure 1.1	(A) Un graphe orienté d'ordre 5 et de taille 6. (B) Un graphe non orienté d'ordre 5 et de taille 4. (C) Un graphe étiqueté, avec des étiquettes sur les arêtes correspondant à des lettres. ...	6
Figure 1.2	Chaîne polynucléotidique d'ARN. Inspiré de (Glouzon, 2017)	11
Figure 1.3	Structure primaire, structure secondaire (fait avec VARNA (Darty <i>et al.</i> , 2009)) et structure tertiaire (fait avec PyMOL (Delano, 2006)) d'un ARN identifié par 1VTQ dans la base de données des acides nucléiques (NBD)(Coimbatore <i>et al.</i> , 2013).	12
Figure 1.4	Les faces d'appariement de l'Adenine (A). Inspiré de (Glouzon, 2017)	13
Figure 1.5	Graphe représentant la structure 3D d'un ARN, les interactions canoniques et non canoniques sont représentées par les figures géométriques de Leontis-Westhof (Leontis et Westhof, 2002), et les interactions d'empilement sont représentées en orange.	16
Figure 1.6	Étant donné deux graphes d'ARN G et H, RIN1 et RIN2 sont des RINs présents dans ces deux graphes. Le RIN2 a deux occurrences, nous verrons plus tard avec les contraintes liées aux RINs que l'arête (5f,7g) sera supprimée du RIN2.	17
Figure 2.1	En considérant les graphes G et H de la figure 1.6, j'obtiens également RIN3, qui est inclus dans RIN1 et contient les mêmes arêtes que RIN1 dans G et H (c'est-à-dire, ils cartographient les mêmes régions de G et H). RIN3 sera supprimé car il représente une fausse variation de RIN1. ...	24
Figure 2.2	Pipeline de recherche des RINs basé sur l'approche PCAM.	28
Figure 2.3	Distribution dynamique des tâches : les points de départ sont divisés en paquets de taille 1. Dès qu'un thread (en jaune) termine l'extension de son point de départ, un autre paquet lui est attribué.....	30
Figure 2.4	Répartition des tâches pour la version imbriquée : lorsqu'un point d'extension est attribué à un thread, celui-ci crée d'autres threads pour partager les tâches d'extension de branche.	32
Figure 2.5	Répartition des tâches pour la version avec vol de travail : lorsqu'un thread n'a plus de tâches à effectuer, il vole les tâches d'extension de branche d'un autre thread.....	34
Figure 3.1	L'évaluation de l'évolutivité pour toutes les versions avec OpenMP. Les deux graphes comparés ont respectivement 390 et 1800 nœuds, et 1003 et 4282 arêtes.	41

Figure 3.2	Le temps d'exécution de 7 comparaisons de graphes sur 10 threads pour toutes les versions avec OpenMP.	41
Figure 3.3	Comparaison des performances (en termes de temps d'exécution et d'accélération) de la version hybride pour différentes allocations $P \times T = 48$, où P est le nombre de processus et T est le nombre de threads. Le numéro sur chaque barre de l'histogramme représente l'accélération.	42
Figure 3.4	Évaluation de l'évolutivité de la version hybride. Les deux graphes comparés ont respectivement 1447 et 1949 nœuds, et 3492 et 4412 arêtes. 1 seul processus est utilisé par nœud.	43
Figure 3.5	Distribution du nombre de nœuds et d'arêtes pour les RINs locaux.	44
Figure 3.6	Distribution de la couverture des RINs locaux.	45
Figure 3.7	Distribution du nombre de nœuds et d'arêtes pour les RINs globaux.	46
Figure 3.8	Répartition de la couverture des RINs globaux.	47
Figure 3.9	Variation des motifs A-MINOR avec empilement. Les lignes orange orientées et non orientées représentent respectivement des interactions d'empilement S53 et S55.	48
Figure 3.10	Variation des motifs Kink-turn U4 snRNA avec empilement En étendant le motif RIN1, nous avons découvert les motifs Kink-turn U4 snRNA et Hm Kt-7.	49
Figure A.1	Distribution du nombre d'arêtes pour l'ensemble de données global à l'échelle logarithmique.	53
Figure A.2	Distribution du nombre d'arêtes pour l'ensemble de données local à l'échelle logarithmique.	54

LISTE DES TABLEAUX

Table 1.1	Les 12 familles d'appariement de paire de bases selon la nomenclature de Leontis-Westhof (Leontis et Westhof, 2002). Inspiré de (Glouzon, 2017).....	14
Table 1.2	Étiquetage des arêtes d'un graphe d'ARN.....	15
Table 3.1	Pour chaque classe, je donne le nombre d'instances (#inst) et ensuite je décris les caractéristiques du graphe : nombre minimum et maximum de nœuds, nombre d'arêtes minimum et maximum.	40

RÉSUMÉ

Les structures complexes des ARN sont essentielles à leurs diverses fonctions. Alors que la structure secondaire organise de manière rigide les tiges, les conformations des boucles sont également importantes et parfois cruciales pour l'architecture globale de la molécule, comme c'est le cas avec les motifs en virage (kink-turn) et A-minor. Les boucles sont formées à partir de réseaux d'interactions de paires de bases, et de nombreux modules récurrents ont été décrits au fil des années. Mettre au jour et organiser ces modules est important pour comprendre la relation séquence-structure-fonction. La nature des molécules d'ARN en tant que réseau d'interactions conduit à leur représentation naturelle sous forme de graphe, avec des arêtes étiquetées représentant des interactions canoniques et non canoniques. Les modules peuvent être considérés comme des sous-graphes communs, des sous-isomorphismes maximaux, entre différentes structures. Trouver ces similitudes automatiquement pose un problème notoirement NP-Difficile, nécessitant d'importantes ressources computationnelles et de mémoire pour être résolu exactement. Différentes approches ont été appliquées, mais ont ignoré les empilements en raison de leur nombre et de leur fréquence élevée dans les structures.

Dans ce mémoire, je présente une méthode distribuée rapide pour extraire des motifs structuraux récurrents présents dans les structures d'ARN, incluant les interactions canoniques, non-canoniques, et pour la première fois toutes les interactions d'empilement. Je fournis des algorithmes et des implémentations efficaces dans le modèle de mémoire partagée ainsi que dans le modèle de mémoire distribuée, avec différentes stratégies de planification des tâches telles que le vol de travail et le maître/ouvrier. Les résultats montrent l'efficacité de la parallélisation avec une bonne évolutivité. J'ai appliqué ma méthode sur toutes les structures d'ARN non-redondantes deux à deux pour trouver tous les modules structurels dans deux contextes : (1) motifs de boucles locaux en ne comparant que des boucles multiples, des boucles intérieures, des renflements et des épingles à cheveux, et (2) motifs globaux où un ensemble de structures entières d'ARN ont été comparé entre elles. L'ensemble de données local contient 16 431 structures d'ARN, j'ai trouvé 631 RINs pour un total de 182 646 occurrences. Et l'ensemble de données global contient 1 722 structures d'ARN, j'ai trouvé 157 344 RINs pour un total de 209 750 474 occurrences. Ces résultats n'étaient pas réalisables avec les travaux précédents, même avec 4 To de mémoire, alors que ma méthode n'a pris que 186 Go de mémoire.

Ma méthode a été implémentée dans un solveur appelé PASIgraph en utilisant le langage de programmation C. J'ai utilisé OpenMP (mémoire partagée) et MPI (mémoire distribuée) pour la parallélisation. Le code source de PASIgraph est accessible via le lien suivant : https://gitlab.info.uqam.ca/agbeto.kossi_wilfried/isomorphisme_graphe

INTRODUCTION

Ces dernières décennies, les structures combinatoires, notamment les graphes, ont connu un intérêt grandissant et constant dans plusieurs domaines tels que les mathématiques, l'informatique, la biologie, la sociologie et d'autres encore. En effet, les graphes, en tant qu'outils abstraits pour représenter les relations entre différentes entités, fournissent un cadre robuste pour la modélisation et la résolution de nombreux problèmes complexes. Par exemple, en biologie, les structures moléculaires telles que les ARN peuvent être représentées sous forme de graphes, où les nœuds représentent les nucléotides et les arêtes représentent les interactions entre eux (Figure 0.1). Cette représentation graphique permet d'analyser les propriétés structurales et fonctionnelles des molécules d'ARN, ainsi que de comprendre les mécanismes sous-jacents impliqués dans les processus biologiques.

0.1 Motifs structuraux d'ARN 3D

L'ARN possède une structure hautement modulaire, caractérisée par des modules essentiels qui façonnent la structure tertiaire de la molécule, déterminant ainsi son rôle fonctionnel crucial au sein de la cellule. Des recherches approfondies (Hendrix *et al.*, 2005; Holbrook, 2005) sur la structure de l'ARN ont montré que les ARN repliés présentent des architectures complexes, qui ressemblent à des assemblages de modules conservés, souvent désignés sous le nom de motifs tertiaires ou motifs structuraux (Djelloul, 2009). Les motifs structuraux se présentent comme des sous-structures ubiquitaires, observées à divers emplacements à l'intérieur de la même molécule d'ARN ou de différentes molécules d'ARN. Ces motifs structuraux jouent un rôle essentiel dans le processus complexe de repliement moléculaire et peuvent également servir de sites de liaison ou d'ancrage pour des protéines ou des ligands (Leontis *et al.*, 2002a; Leontis et Westhof, 2003; Lescoute *et al.*, 2005a). L'identification et l'étude des motifs structuraux sont donc cruciales pour comprendre la structure et la fonction de l'ARN.

Les motifs structuraux peuvent être classés en deux grandes catégories :

1. **Motifs locaux** : Ces motifs sont insérés dans les éléments de la structure secondaire (Boucles, tiges). Parmi les motifs locaux bien connus dans les molécules d'ARN, on trouve les Kink-turns (Huang et Lilley, 2016) et les G-bulges (Hermann et Patel, 2000).
2. **Motifs d'interactions** : Ils se forment par des interactions entre 2 ou plusieurs éléments de la structure secondaire. Ces motifs sont largement moins étudiés que les motifs locaux. Le seul motif bien connu de cette catégorie est le A-minor (Nissen *et al.*, 2001).

Structure tertiaire

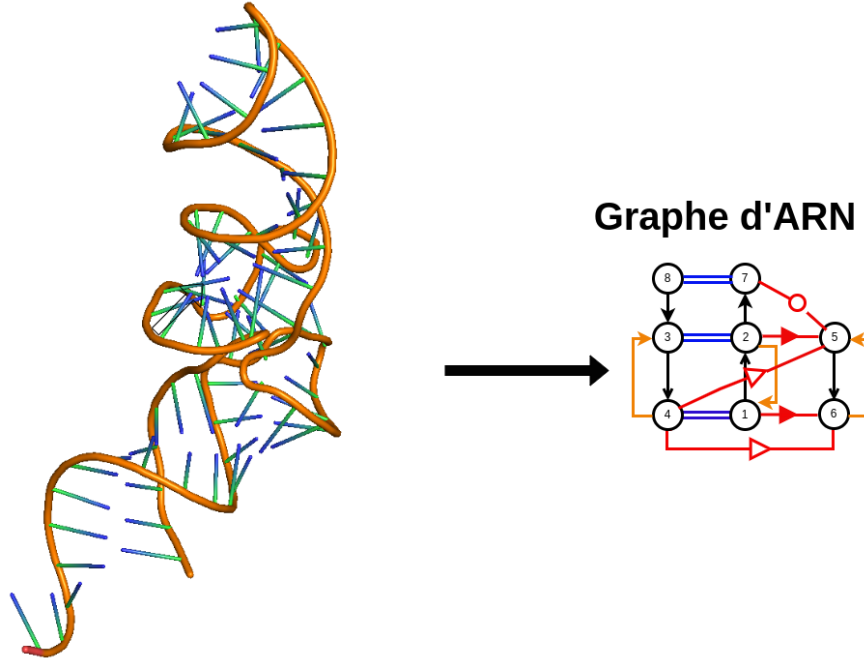


Figure 0.1 Structure tertiaire d'un ARN et sa représentation sous forme de graphe. Les interactions en bleu sont locales à un élément de la structure secondaire, et les interactions en rouge sont à longue portée.

Ce mémoire se concentre sur les réseaux d'interaction récurrents (RIN), qui sont des motifs structuraux ne contenant aucune information de séquence, mais uniquement des informations topologiques sur les interactions entre les nucléotides et la nature de ces interactions. Lorsqu'il est associé à des informations de séquence, un RIN peut donner lieu à un ou plusieurs motifs structuraux.

0.2 Problème d'extraction des motifs structuraux

Les méthodes automatisées d'extraction de motifs structuraux dans les ARN 3D peuvent être classées en deux catégories principales. **Les méthodes géométriques (Wang et al., 2007; Wadley et Pyle, 2004; Petrov et al., 2013)** se concentrent sur la géométrie tridimensionnelle des structures d'ARN pour identifier et classifier les motifs. Elles utilisent des propriétés géométriques telles que les angles de torsion du squelette ou les coordonnées cartésiennes des atomes. Par exemple, RNA 3D Motif Atlas (Petrov et al., 2013) est une base de données contenant des motifs locaux tels que les boucles internes et les boucles en épingles à cheveux de l'ARN 3D. Pour identifier ces motifs, RNA 3D Motif Atlas utilise la suite logicielle FRED (Sarver et al., 2007) pour aligner des boucles de l'ARN 3D extraite de la PDB. **Les méthodes de la théorie des graphes**

représentent les structures d'ARN sous forme de graphes, où les nœuds représentent les nucléotides et les arêtes représentent les interactions entre eux. Ces méthodes sont souvent utilisées pour retrouver des occurrences de motifs connus dans de nouvelles structures d'ARN. Certaines de ces méthodes (Djelloul, 2009; Zhong et Zhang, 2011) recherchent des motifs sans aucune connaissance préalable de leur géométrie ou topologie, mais se concentrent uniquement sur les motifs locaux.

À ma connaissance, seuls deux projets se sont intéressés au calcul exhaustif des RINs. Le premier (Reinharz *et al.*, 2018) se concentre uniquement sur la recherche des RINs de motifs d'interactions et limite le nombre d'éléments de la structure secondaire impliqués dans ces motifs. Le second (Soulé *et al.*, 2021) va au-delà des limites du premier, mais recherche des RINs dans la structure secondaire. De plus, la méthodologie proposée par ces deux projets prend beaucoup de temps pour la recherche des RINs, surtout pour de grandes structures d'ARN.

0.3 Contribution

Ce mémoire contribue à la recherche automatisée *de novo* de motifs structuraux basés sur leurs interactions plutôt que sur leur séquence ou leur contexte. Contrairement aux autres études sur les RINs, mon approche permet d'identifier à la fois des motifs locaux et des motifs d'interaction dans la structure tertiaire.

La méthode proposée utilise une représentation en graphe de l'ARN, où les nœuds représentent les nucléotides identifiés par leur nom et leur numéro de séquence, et les arêtes représentent les interactions entre ces nucléotides, étiquetées par leurs différents types. En se basant sur ce modèle de graphe d'ARN, les RINs peuvent être considérés comme des sous-graphes communs. J'ai développé un algorithme pour trouver tous les MCCPS (Sous-graphes Partiels Communs Connexes Maximaux) entre deux graphes, qui a été utilisé pour comparer par paire toutes les structures d'ARN non redondantes afin de retrouver tous les RINs qu'elles contiennent.

Le problème de retrouver tous les MCCPS est une généralisation du problème du sous-graphe commun maximum, reconnu dans la littérature comme étant NP-difficile (Cook, 1971). Afin d'améliorer les performances de l'algorithme, j'ai utilisé la programmation parallèle. Les performances obtenues m'ont permis d'incorporer pour la première fois les interactions d'empilement dans la recherche de RINs.

0.4 Structure du document

Le premier chapitre introduit les notions informatiques et biologiques nécessaires à la bonne compréhension du mémoire. J'y décris la structure de l'ARN et les motifs structuraux. Le modèle de graphe d'ARN utilisé y est présenté, de même que la définition de la notion de RINs. Le deuxième chapitre présente la méthodologie utilisée pour l'identification des RINs. Ensuite, je présente les résultats obtenus avec une évaluation de la performance de l'algorithme de recherche des RINs. Pour finir, une conclusion avec des travaux futurs est proposée.

CHAPITRE 1

ARN

L'ARN (acide ribonucléique) est une molécule biologique essentielle trouvée dans tous les organismes vivants. Son rôle principal est de transmettre l'information génétique contenue dans l'ADN et de la traduire en protéines fonctionnelles lors du processus de synthèse des protéines. La structure de l'ARN est cruciale pour sa fonction biologique, car elle détermine sa capacité à interagir avec d'autres molécules et à effectuer diverses fonctions cellulaires. De ce fait, il est important d'étudier les structures de l'ARN afin de mieux comprendre la relation structure-fonction. L'étude des motifs structuraux nécessite la capacité de modéliser la structure tertiaire de l'ARN. Ce modèle doit être capable de capturer toutes les informations biologiques inhérentes à la structure tertiaire de l'ARN. Dans ce chapitre, je vais décrire le cadre théorique nécessaire à la compréhension du modèle de graphe d'ARN. Ensuite, j'introduirai les notions biologiques nécessaires à la bonne compréhension du mémoire. Nous explorerons en détail la structure de l'ARN, y compris les notions nécessaires à la compréhension des motifs structuraux. Pour finir je présenterai le modèle de graphe d'ARN utilisé, où nous verrons que les motifs structuraux récurrents se réfèrent à des sous-graphes partiels communs connexes maximaux (MCCPS).

1.1 Modélisation informatique

1.1.1 Les Graphes

Un **graphe** G est décrit par un ensemble V de sommets et un ensemble d'arêtes $E \subseteq V \times V$, où chaque arête est une paire de sommets. On appelle sommets **adjacents** des paires de sommets qui sont reliés par une arête. L'**ordre** d'un graphe correspond au nombre de sommets qu'il contient, tandis que sa **taille** correspond à son nombre d'arêtes. Les sommets, les arêtes, ou les deux peuvent être associés à des étiquettes ou des valeurs représentant diverses informations telles que des noms, des poids, des attributs ou des catégories. On parle ainsi de **graphe étiqueté**. Les graphes peuvent être aussi **orientés** ou **non orientés**.

Soit a et $b \in V$. Dans un graphe non orienté, les arêtes sont **symétriques**. Une arête $\{a, b\}$ représente une **relation symétrique (bidirectionnelle)** entre les sommets a et b . En revanche, dans un graphe orienté, les arêtes sont **orientées**. Une arête ou arc (a, b) indique une **relation unidirectionnelle** allant du sommet a vers le sommet b . L'arête (b, a) est appelée **arête inverse** de (a, b) .

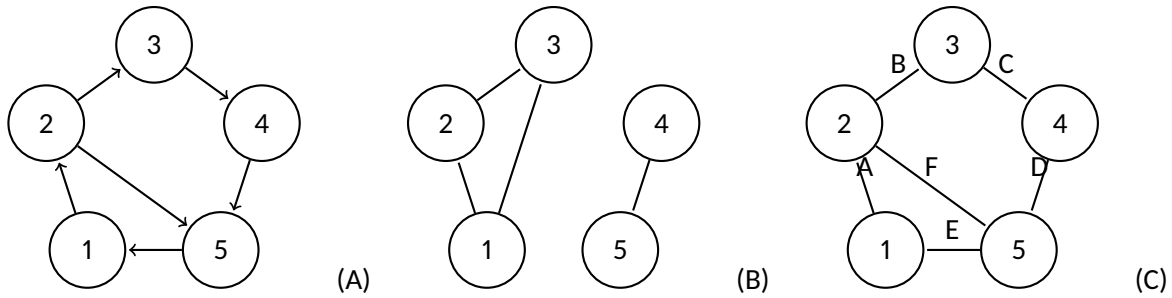


Figure 1.1 (A) Un graphe orienté d'ordre 5 et de taille 6. (B) Un graphe non orienté d'ordre 5 et de taille 4. (C) Un graphe étiqueté, avec des étiquettes sur les arêtes correspondant à des lettres.

1.1.1.1 Terminologie des graphes

Dans les définitions suivantes, je me place dans le cadre de graphes orientés. Toutefois, ces notions peuvent être étendues aux graphes non orientés.

- Un **graphe connexe** est un graphe dans lequel il existe un **chemin** (séquence de sommets où chaque paire de sommets consécutifs est adjacente) reliant chaque paire de nœuds (Figure 1.1 graphe A).
- Un **graphe non-connexe** est un graphe constitué de plusieurs composants connexes (Figure 1.1 graphe B).
- Un **multidigraphe** est un graphe orienté dans lequel il peut exister plusieurs arcs ou arêtes orientées entre la même paire de sommets.
- Deux graphes G et H sont dits **isomorphes** s'il existe une bijection $f : V_G \rightarrow V_H$ entre les ensembles de sommets de G et H , telle que pour chaque paire de sommets a et b dans G , l'arête (a, b) est présente dans G si et seulement si l'arête $(f(a), f(b))$ est présente dans H . Autrement dit, deux graphes sont isomorphes s'ils ont le même ordre et qu'ils sont connectés de la même manière.
- Un **sous-graphe partiel** G' d'un graphe G est obtenu en retirant une ou plusieurs arêtes de G . Plus précisément, $G' = (V', E')$ est un sous-graphe partiel de $G = (V, E)$ si et seulement si E' est un sous-ensemble de E .
- Un **sous-graphe induit** $G' = (V_{G'}, E_{G'})$ d'un graphe $G = (V, E)$ est tel que $V_{G'} \subseteq V$ et $E_{G'} = E(V_{G'})$, où $E(V_{G'})$ est l'ensemble des arêtes de G dont les deux extrémités sont dans $V_{G'}$. Cela signifie que G' est formé en conservant un sous-ensemble de sommets $V_{G'}$ de G , ainsi que toutes les arêtes de G qui ont leurs deux extrémités dans $V_{G'}$.
- Un **sous-graphe commun** H de deux graphes G et G' est un graphe qui est isomorphe à des sous-graphes de G et G' .
- Un **sous-graphe commun maximum** peut être défini de deux manières :

1. **Sous-graphe Partiel Commun Maximum (MCPS - Maximum Common Partial Subgraph)** : Il s'agit d'un sous-graphe commun qui contient le plus grand nombre d'arêtes parmi tous les sous-graphes communs de deux graphes donnés.

2. **Sous-graphe Induit Commun Maximum (MCIS - Maximum Common Induced Subgraph)** : Il s'agit d'un sous-graphe commun qui contient le plus grand nombre de sommets parmi tous les sous-graphes communs de deux graphes donnés.

- Un **sous-graphe commun maximal** de deux graphes G et G' est un sous-graphe qui ne peut pas être étendu (c'est-à-dire auquel on ne peut pas ajouter de sommet ou d'arête) sans perdre sa propriété d'être isomorphe à un sous-graphe à la fois de G et de G' . En d'autres termes, il s'agit d'un sous-graphe commun qui est aussi grand que possible tout en restant isomorphe à un sous-graphe de G et de G' .

1.1.2 Le problème de trouver tous les MCCPS entre 2 graphes

Le problème de trouver les sous-graphes partiels communs connexes maximaux (MCCPS) est une généralisation qui inclut la connectivité du problème du MCPS (Maximum Common Partial Subgraph), qui se limite à trouver uniquement le plus grand sous-graphe partiel commun maximal entre deux graphes.

1.1.2.1 État de l'art

Le problème de trouver un sous-graphe commun maximal (MCS) a été étudié depuis un demi-siècle. Le problème de trouver un MCS est divisé en deux catégories :

1. MCIS (Maximum Common Induced Subgraph) (Ndiaye et Solnon, 2011; Quer *et al.*, 2020; Schmidt *et al.*, 2022), est le problème de trouver un sous-graphe induit commun avec le plus grand nombre de nœuds.
2. MCPS (Maximum Common Partial Subgraph) (Bahiense *et al.*, 2012; Raymond *et al.*, 2002), est le problème de trouver un sous-graphe partiel commun avec le plus grand nombre d'arêtes.

Une autre différenciation peut être faite entre le cas connexes et le cas non connexes. (Kann, 1992; Huang *et al.*, 2006) ont démontré que ce problème est l'un des plus difficiles à résoudre en termes de complexité algorithmique.

Il existe deux principaux types d'approches exactes pour résoudre ce problème : l'approche de réduction au

problème du clique maximum et l'approche d'énumération des sous-graphes communs (branch and bound) (Minot *et al.*, 2015). L'approche de réduction au problème du clique maximum repose sur une reformulation du problème dans un graphe de compatibilité. Dans ce graphe, chaque clique correspond à un sous-graphe commun. Ainsi, trouver un sous-graphe commun maximum revient à identifier une clique maximum dans le graphe de compatibilité. Dans l'approche d'énumération des sous-graphes communs, un arbre de recherche est construit pour explorer tous les sous-graphes communs possibles. La création des sous-graphes est arrêtée lorsque l'algorithme détermine qu'un sous-graphe ne peut pas produire de solution plus grande que la meilleure trouvée jusqu'à présent. L'efficacité de ces algorithmes dépend de l'utilisation d'heuristiques d'élagage pour supprimer rapidement les branches inutiles de l'arbre de recherche (Minot *et al.*, 2015). La deuxième approche est plus efficace pour les graphes de petite taille ou peu denses. Dans tous les autres cas, la première approche est plus efficace (Conte *et al.*, 2007).

Des approches approximatives (Raymond et Willett, 2002) et parallèles (McCreesh et Prosser, 2013; Depolli *et al.*, 2013) ont également été proposées. Les approches parallèles décomposent le problème en sous-problèmes indépendants qui peuvent être résolus simultanément en utilisant une partition de domaine. Le problème MCS est l'un des problèmes les plus difficiles à paralléliser en raison de sa nature combinatoire complexe et de l'interdépendance entre les sous-problèmes.

Un autre exemple d'applications parallèles irrégulières est l'exploration de l'espace des paramètres. L'espace des paramètres est découpé à partir de points appelés *points de référence*. À partir d'un point de référence, sont calculés les polyèdres autour de celui-ci. Différents polyèdres peuvent être calculés en parallèle car leur calcul est indépendant les uns des autres. Cependant, il est impossible de savoir à l'avance quels points seront inclus dans le même polyèdre. Il est donc important d'éviter de calculer le même polyèdre sur différentes ressources informatiques. Des heuristiques et des algorithmes dynamiques ont été conçus et évalués dans (André *et al.*, 2015a; André *et al.*, 2015b). Leur conclusion est que pour de petits espaces de paramètres, une distribution aléatoire des points fonctionne bien, et sur de plus grands espaces de paramètres, une décomposition de domaine dynamique avec vol de travail fonctionne mieux. (Coti *et al.*, 2019) un problème de découpage d'espace de paramètres comparable. Ils comparent différentes approches, et celle qui fonctionne le mieux consiste à détecter et à arrêter les calculs redondants tôt et à calculer une autre tâche à la place.

1.1.2.2 Défis

- Complexité combinatoire : Trouver les MCCPS implique une exploration exhaustive de l'espace de recherche. Le nombre de sous-graphes possibles augmente de manière exponentielle avec la taille des graphes.
- Dépendance entre les sous-problèmes : Les MCCPS sont interdépendants. Ajouter ou supprimer un sommet ou une arête d'un sous-graphe peut affecter la validité d'autres sous-graphes (travailler sur des sous-graphes qui ont déjà été trouvés par d'autres tâches parallèles). Par conséquent, la parallélisation de ces sous-problèmes peut entraîner des conflits et des problèmes de synchronisation, rendant difficile la conception d'algorithmes parallèles efficaces.
- Déséquilibre de charge : Les MCCPS ne sont pas nécessairement de la même taille ou complexité. Certains sous-problèmes peuvent être résolus rapidement, tandis que d'autres peuvent nécessiter beaucoup plus de temps. Cela peut entraîner une distribution inégale de la charge de travail entre les unités d'exécution parallèles.

Dans la prochaine section, je décrirai plus en détail la structure de l'ARN ainsi que les motifs structuraux. Nous examinerons également le modèle de graphe utilisé pour représenter ces structures.

1.2 Structure de l'ARN

Contrairement à l'ADN, qui se présente sous forme de double brin (bicaténaire) formant une double hélice, l'ARN est généralement simple brin (monocaténaire) et est constitué d'une chaîne de nucléotides plus courte. L'ARN possède trois niveaux structuraux : la structure primaire, la structure secondaire et la structure tertiaire.

La structure primaire de l'ARN (Figure 1.2) est constituée de nucléotides qui sont reliés entre elles par des liaisons phosphodiester formant ainsi une chaîne de nucléotides orientée de l'extrémité 5' vers l'extrémité 3'. Les nucléotides sont constitués de trois éléments principaux : un groupe phosphate, un sucre (ribose) et une base azotée. Dans l'ARN, il existe quatre bases nucléiques : l'adénine, la cytosine, la guanine et l'uracile. Chacune de ces bases est respectivement représentée par les lettres : A, C, G et U. On distingue deux groupes de bases dans l'ARN : les pyrimidines (C et U) et les purines (A et G). Les pyrimidines sont composées d'une seule structure cyclique, tandis que les purines sont composées de deux structures cycliques. La séquence nucléotidique de l'ARN se réfère à l'ordre de succession des nucléotides. Cette séquence est déterminante pour la fonction biologique de l'ARN, car elle code l'information génétique ou contient des éléments régulateurs qui contrôlent divers processus cellulaires. Les bases de l'ARN interagissent de manière complémentaire entre elles. L'adénine forme des liaisons avec l'uracile, tandis que la guanine forme des liaisons avec la cytosine. Cette complémentarité des bases est importante pour la formation de structures secondaires de l'ARN.

En raison de sa nature simple brin, l'ARN est capable de se replier sur elle-même en formant des appariements entre les bases nucléiques complémentaires comme mentionné précédemment, pour former successivement la structure secondaire et la structure tertiaire (Figure 1.3). Ces structures sont essentielles pour comprendre les rôles et les fonctions de l'ARN au sein de la cellule. L'un des défis majeurs de la bio-informatique réside dans la capacité à prédire avec précision la structure secondaire et tertiaire des ARN. L'étude des motifs structuraux représente ainsi un grand pas vers la réalisation de cet objectif.

Pendant le processus de repliement de l'ARN, les appariements entre les paires de nucléotides peuvent varier en fonction des éléments chimiques impliqués de part et d'autre de l'appariement. Ces éléments chimiques sont : la base, le sucre ou le phosphate. En ce qui concerne les appariements base-base, on distingue 2 types principaux : les appariements côté-côté et les empilements (Zirbel *et al.*, 2009; Djelloul, 2009).

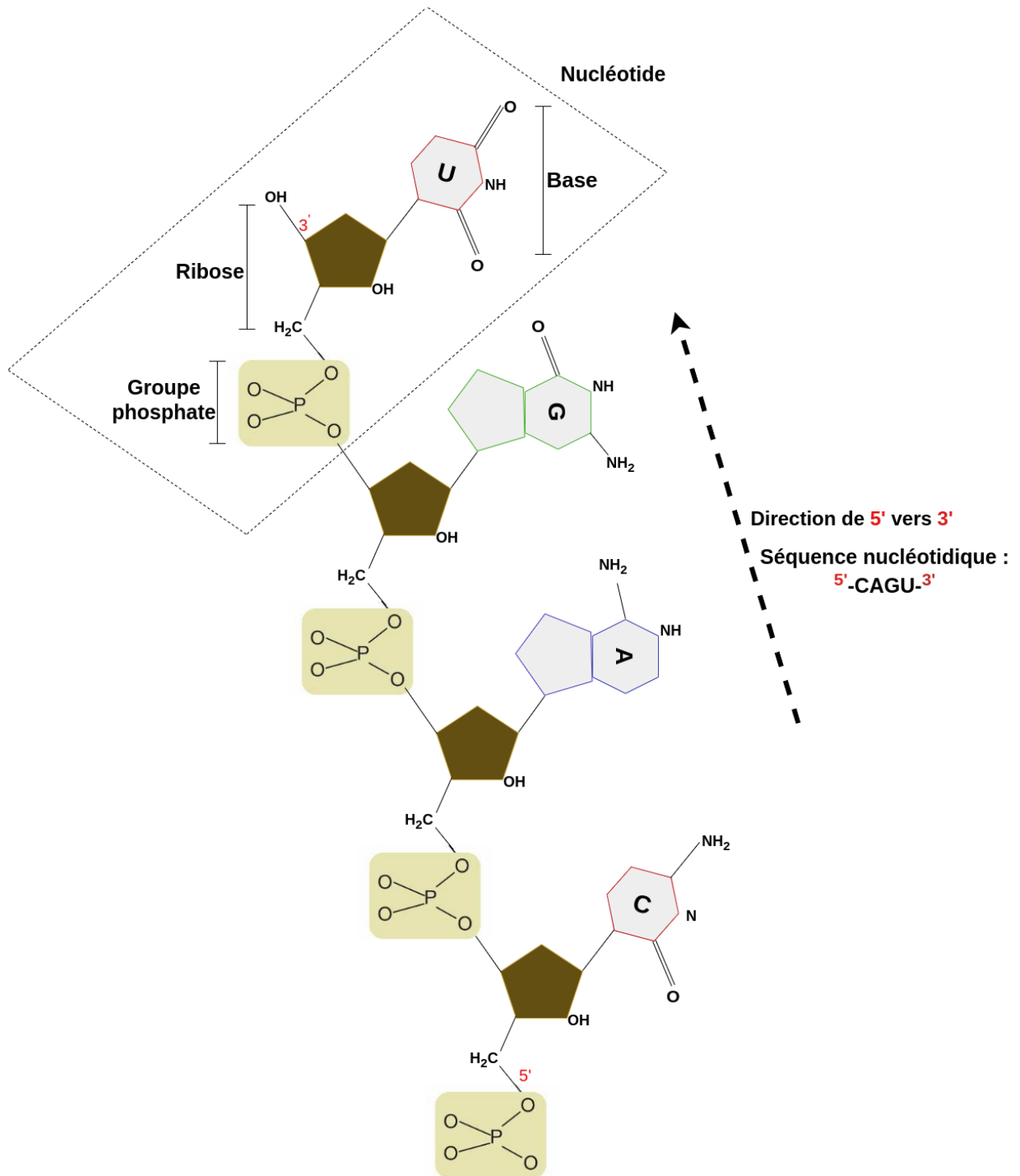
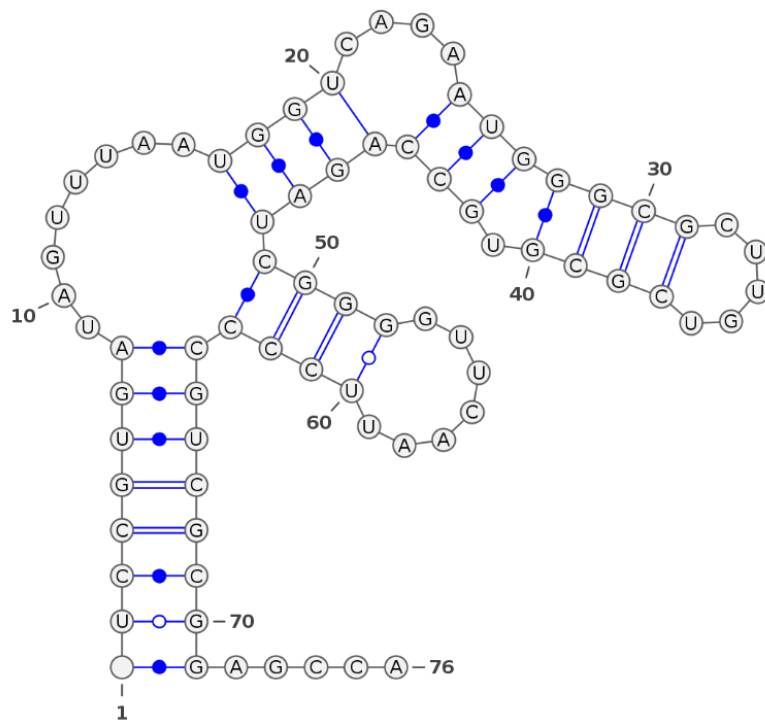


Figure 1.2 Chaîne polynucléotidique d'ARN. Inspiré de (Glouzon, 2017)

Structure primaire

5'UCCGUGAUAGUUUAAUGGUCAGAAUGGGCGCUUGUCG
CGUGCCAGAUCGGGGUUCAAUUCCCCGUCGCGGAGCCA3'

Structure secondaire



Structure tertiaire

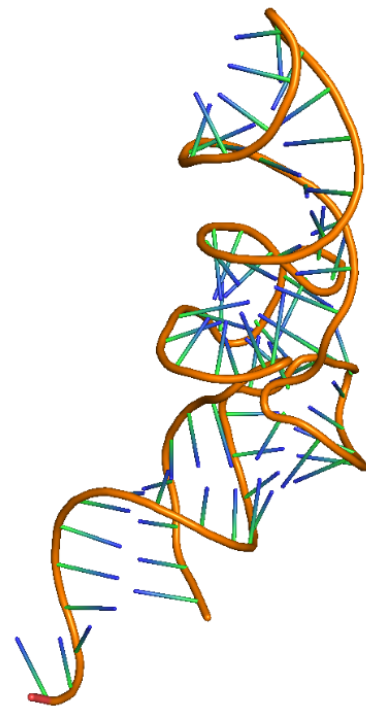


Figure 1.3 Structure primaire, structure secondaire (fait avec VARNA (Darty *et al.*, 2009)) et structure tertiaire (fait avec PyMOL (Delano, 2006)) d'un ARN identifié par 1VTQ dans la base de données des acides nucléiques (NBD)(Coimbatore *et al.*, 2013).

Les appariements côté-côté sont établis à l'aide de liaisons hydrogène et sont classés en 12 familles selon la nomenclature de Leontis-Westhof (Tableau 1.1) en fonction du côté et de l'orientation des bases impliquées. Le côté d'appariement des bases fait référence à la face de chaque base nucléique impliquée dans l'appariement. Il existe trois faces d'appariement (Figure 1.4) : Watson-Crick (W), Hoogsteen (H) et Sugar (S). Il existe deux types d'orientation des bases appariées : Cis et trans, qui font référence à l'orientation des liaisons glycosidiques (liaison entre le ribose et le groupe phosphate). Dans une interaction en cis, les liaisons glycosidiques des bases appariées se situent dans le même plan, ce qui signifie qu'elles se trouvent du même côté du brin d'ARN. En revanche, dans une orientation en trans, les liaisons glycosidiques des nucléotides appariés ne se situent pas dans le même plan, ce qui signifie qu'elles se trouvent de part et d'autre du brin d'ARN (Leontis *et al.*, 2002c; Leontis *et al.*, 2002b). Les interactions ou appariements canoniques désignent les appariements Watson-Crick (W/W) en cis entre la cytosine (C) et la guanine (G), ainsi qu'entre l'adénine (A) et l'uracile (U). En plus de ces appariements canoniques, on peut également observer des appariements entre la guanine (G) et l'uracile (U), appelés appariements wobble (Leontis *et al.*, 2002c; Leontis *et al.*, 2002b). Le reste des appariements côté-côté est appelé interactions non canoniques.

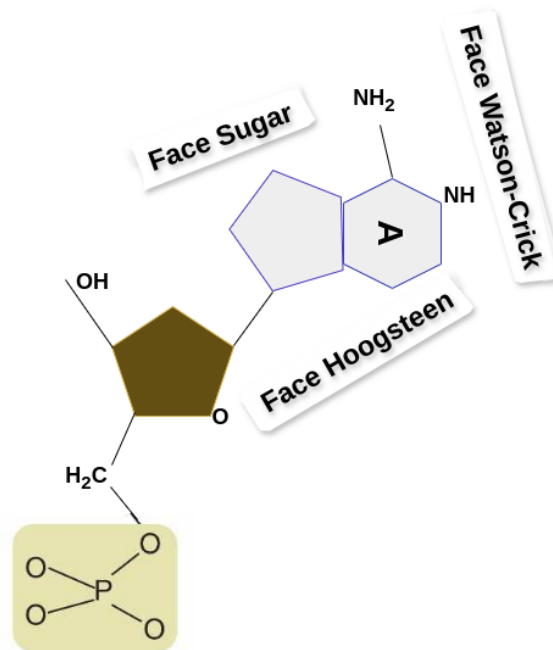


Figure 1.4 Les faces d'appariement de l'Adénine (A). Inspiré de (Glouzon, 2017)

La structure secondaire de l'ARN est principalement déterminée par les interactions canoniques et wobble entre les paires de bases, formant ainsi des tiges, tandis que les nucléotides non appariés forment des boucles. Les appariements non canoniques ou les interactions d'empilement s'établissent ensuite entre les






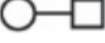



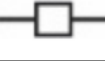



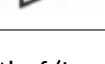
Orientation de la liaison glycosidique	Face d'interaction	Symbole
Cis	Watson-Crick/Watson-Crick (CWW)	
Cis	Watson-Crick/Watson-Crick (CWW) (<i>G - C</i>) ¹	
Cis	Watson-Crick/Watson-Crick (CWW) (<i>A - U</i>) ²	
Trans	Watson-Crick/Watson-Crick (TWW)	
Cis	Watson-Crick/Hoogsteen (CWH)	
Trans	Watson-Crick/Hoogsteen (TWH)	
Cis	Watson-Crick/Sugar (CWS)	
Trans	Watson-Crick/Sugar (TWS)	
Cis	Hoogsteen/Hoogsteen (CHH)	
Trans	Hoogsteen/Hoogsteen (THH)	
Cis	Hoogsteen/Sugar (CHS)	
Trans	Hoogsteen/Sugar (THS)	
Cis	Sugar/Sugar (CSS)	
Trans	Sugar/Sugar (TSS)	

Table 1.1 Les 12 familles d'appariement de paire de bases selon la nomenclature de Leontis-Westhof (Leontis et Westhof, 2002). Inspiré de (Glouzon, 2017)

nucléotides non appariés pour former la structure tertiaire (Sarver *et al.*, 2008), qui détermine l'ensemble des fonctions de l'ARN. Les appariements canoniques et non canoniques qui déterminent la structure tertiaire s'organisent de manière modulaire, formant différents motifs appelés motifs structuraux (Leontis *et al.*, 2006; Shalybkova *et al.*, 2021).

(Sarver *et al.*, 2008) a effectué un classement des interactions d'empilement, visant à décrire les différentes faces des bases appariées en fonction de leur orientation dans la molécule d'ARN. La face qui est orientée vers l'extrémité 3' est désignée par Face 3', tandis que la face orientée vers l'extrémité 5' est désignée par Face 5'. Ainsi on peut avoir des interactions d'empilement 35, 33 et 55

Les motifs structuraux sont considérés comme récurrents lorsqu'ils se retrouvent à des endroits non homologues du même ARN ou de différents ARN et qu'ils possèdent une structure 3D similaire (Sarver *et al.*, 2008). La présence de motifs structuraux récurrents est importante car elle suggère une fonction biologique spécifique associée à ces structures. Ces motifs peuvent jouer un rôle dans la régulation génique, et peuvent être également des sites de liaison pour des protéines ou des ligands. Leur récurrence dans différentes molécules d'ARN suggère qu'ils sont conservés au cours de l'évolution en raison de leur importance fonctionnelle. L'identification et l'étude des motifs structuraux récurrents sont donc cruciales pour comprendre la structure et la fonction de l'ARN, ainsi que pour élucider les mécanismes moléculaires sous-jacents à divers processus biologiques.

1.2.1 L'ARN comme graphe

La structure 3D de l'ARN peut être modélisée sous forme de graphe, où chaque sommet représente un nucléotide identifié par son nom de base et son numéro de séquence, et où les arêtes représentent les interactions entre les nucléotides, étiquetées par leur type d'interaction.

Table 1.2 Étiquetage des arêtes d'un graphe d'ARN.

Edge type	Edge label	inverse edge label
BackBone	B53	B35
Interactions canoniques et non canoniques	$\alpha\beta\gamma$, $\alpha \in \{T,C\}$, β et $\gamma \in \{W,H,S\}$	$\alpha\gamma\beta$
Empilement	SAB, A et B $\in \{5,3\}$	SBA

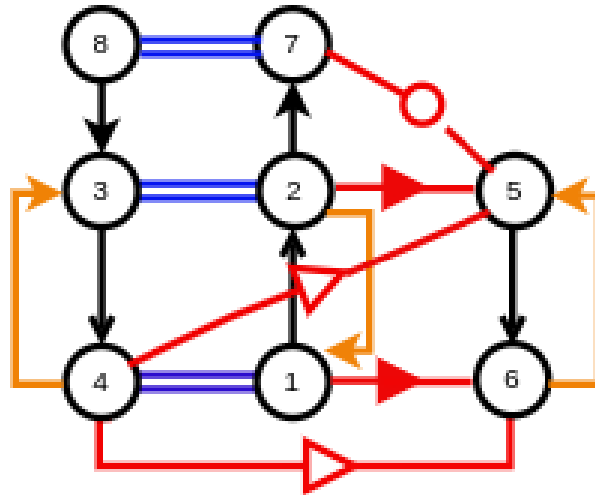


Figure 1.5 Graphe représentant la structure 3D d'un ARN, les interactions canoniques et non canoniques sont représentées par les figures géométriques de Leontis-Westhof (Leontis et Westhof, 2002), et les interactions d'empilement sont représentées en orange.

Mon modèle utilise un multigraphe orienté (Multidigraphe) pour représenter la structure de l'ARN. Le choix d'utiliser un multigraphe est justifié par le fait qu'il peut exister plusieurs interactions entre les mêmes paires de nucléotides. Par exemple, une paire de nucléotides peut être impliquée à la fois dans une interaction de backbone et dans une interaction d'empilement (Figure 1.5 sommets 1 et 2). Le choix de rendre le multigraphe orienté est justifié par le fait que toutes les interactions ne sont pas symétriques. Par exemple, une interaction d'appariement Cis Hoogsteen Sugar (CHS) entre deux nucléotides n'est pas symétrique car les faces d'interaction des nucléotides sont différentes.

1.2.2 RIN (Réseaux d'interaction récurrents)

Comme l'a dit (Lescoute *et al.*, 2005b), les occurrences des motifs structuraux peuvent être différentes au niveau de l'identité des paires de bases impliquées dans les interactions d'appariement. De plus, certaines études (Gianfrotta, 2022) ont montré que la conservation du contexte topologique sur les interactions entre nucléotides et la nature de ces interactions peut impliquer la conservation du contexte 3D. Dans ce sens, nous définissons la notion de RIN (Réseaux d'Interactions Récurrents).

Les RINs sont des motifs structuraux ne contenant aucune information de séquence, mais uniquement des informations topologiques sur les interactions entre les nucléotides et la nature de ces interactions. Lorsqu'il est associé à des informations de séquence, un RIN peut donner lieu à un ou plusieurs motifs structuraux.

En se plaçant dans le modèle de graphe d'ARN, les motifs structuraux récurrents (structure se retrouvant à des endroits non homologues du même ARN ou de différents ARN) peuvent être interprétés comme des sous-graphes communs connexes maximaux. Quant aux RINs, qui sont des représentations des motifs structuraux récurrents avec uniquement des informations topologiques sur les interactions entre les nucléotides, ils peuvent être identifiés comme des sous-graphes partiels communs connexes maximaux (MCCPS-Maximal Connected Common Partial Subgraphs). Ce que je cherche à faire dépasse les capacités des algorithmes proposés par les travaux précédents. Par conséquent, j'ai développé un algorithme pour trouver tous les MCCPS entre deux graphes, qui a été utilisé pour effectuer une comparaison par paire de toutes les structures d'ARN non redondantes afin de retrouver tous les RINs qu'elles contiennent.

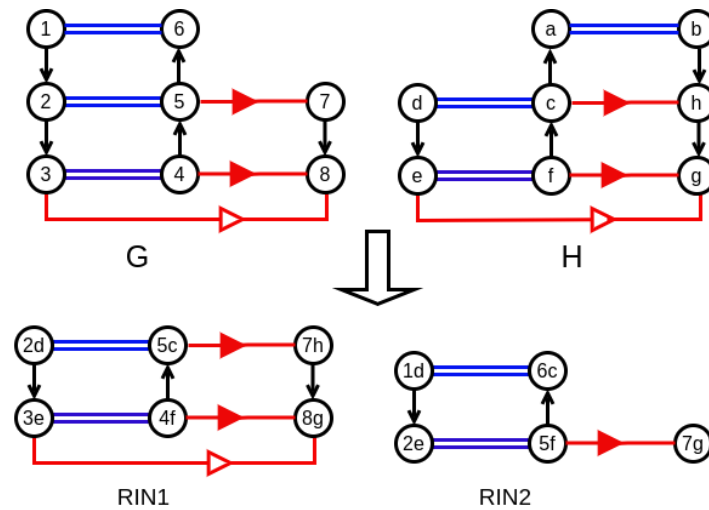


Figure 1.6 Étant donné deux graphes d'ARN G et H, RIN1 et RIN2 sont des RINs présents dans ces deux graphes. Le RIN2 a deux occurrences, nous verrons plus tard avec les contraintes liées aux RINs que l'arête (5f,7g) sera supprimée du RIN2.

CHAPITRE 2

MÉTHODOLOGIE

2.1 Algorithme pour les Sous-graphes Partiels Communs Connexes Maximaux (MCCPS)

Soient g et h deux graphes dirigés avec des étiquettes sur les arêtes. L'approche utilisée pour trouver tous les **MCCPS** entre g et h consiste à partir d'une correspondance d'arêtes entre g et h et à ajouter une correspondance d'arêtes voisines à celle-ci, en considérant toutes les possibilités jusqu'à atteindre le maximal.

Cette méthode contient les tâches suivantes :

1. Recherche de toutes les correspondances d'arêtes ;
2. Extension des correspondances d'arêtes ;

2.1.1 Recherche de toutes les correspondances d'arêtes

Une arête de g et une arête de h correspondent ensemble si elles ont la même étiquette et la même direction. J'utilise une liste de description des types d'arêtes (Tableau 1.2) pour traiter une correspondance d'arêtes et sa correspondance d'arêtes inverses comme une seule correspondance d'arêtes. Par exemple, pour $edgeTypeList = \{ \{CWS, CSW\}, \{TWW, TWW\} \}$. Le premier élément de $edgeTypeList$ implique que s'il existe dans g une arête (a, b, CWS) , alors il existe également dans g une arête inverse (b, a, CSW) .

La fonction `SearchAllEdgeMatches` (Algorithme 1) retourne la liste de toutes les correspondances d'arêtes trouvées (**edgeMatchList**) ainsi que deux autres listes :

- **edgeMatchIdList1** : Elle contient l'identifiant de l'arête dans g pour chaque correspondance d'arêtes ;
- **edgeMatchIdList2** : Elle contient l'identifiant de l'arête dans h pour chaque correspondance d'arêtes ;

Algorithme 1: SearchAllEdgeMatches

Data: $g, h, \text{edgeTypeList}$

```
1 edgeMatchList  $\leftarrow$  {};  
2 edgeMatchIdList1  $\leftarrow$  {};  
3 edgeMatchIdList2  $\leftarrow$  {};  
4 for  $i \leftarrow 0$  to  $\text{edgeTypeList.size}()$  do  
5     edgeType  $\leftarrow$   $\text{edgeTypeList}[i]$ ;  
6     label  $\leftarrow$   $\text{edgeType.edgeLabel}$ ;  
7     edgeList1  $\leftarrow$   $\text{getOutgoingEdge}(g, \text{label})$ ;  
8     edgeList2  $\leftarrow$   $\text{getOutgoingEdge}(h, \text{label})$ ;  
9     for  $\text{edge1}$  in  $\text{edgeList1}$  do  
10        for  $\text{edge2}$  in  $\text{edgeList2}$  do  
11            matchEdge  $\leftarrow$   $\text{createMatchEdge}(\text{edge1}, \text{edge2})$ ;  
12            edgeMatchList.append(matchEdge);  
13            edgeMatchIdList1.append(edge1.id);  
14            edgeMatchIdList2.append(edge2.id);
```

2.1.2 Extension des correspondances d'arêtes

Les correspondances d'arêtes sont les motifs élémentaires constituant les MCCPS. Dans cette partie, je présente l'algorithme qui consiste à étendre une correspondance d'arêtes pour trouver tous les MCCPS qui la contiennent.

Point de départ : (Algorithme 2 ligne 4) Un point de départ s pour deux graphes g et h est une correspondance d'arêtes entre g et h . En d'autres termes, l'ensemble des points de départ est l'ensemble des correspondances d'arêtes.

Extension à partir d'un point de départ s : (Algorithme 4) Il s'agit de trouver un MCCPS en effectuant une recherche en largeur d'abord (BFS) à partir du point de départ s . Le sous-graphe est construit dynamiquement en ajoutant au point de départ s des correspondances d'arêtes voisines jusqu'à maximisation. La maximisation est atteinte lorsqu'il n'y a plus de voisins à explorer. Une correspondance d'arêtes est ajoutée au sous-graphe en cours construction seulement lorsqu'elle a un nœud qui est déjà présent dans le sous-graphe, ce qui est nécessaire pour maintenir la connectivité.

Sous-graphe partiel commun connexe (PCCS) : C'est un sous-graphe connexe en cours de construction à partir d'un point de départ s et qui n'est pas encore maximal.

Conflits : (Algorithme 4 ligne 9 ; ligne 15-17) Lors de l'extension à partir d'un point de départ s , des événements de conflit peuvent survenir. Un événement de conflit se produit lorsqu'une correspondance d'arêtes voisines ne peut pas être ajoutée au PCCS. Cela se produit lorsqu'une correspondance d'arêtes voisines a un nœud $a_g | a_h$ qui ne peut pas être ajouté au PCCS, car ce dernier a soit un nœud $a'_g | a_h$ soit un nœud $a_g | a'_h$ ou les deux (avec $a'_g \neq a_g$ et $a'_h \neq a_h$). Un conflit suggère l'existence d'un MCCPS différent de celui en cours de production. En effet, un conflit indique l'existence d'un MCCPS contenant au moins une correspondance d'arêtes (la correspondance d'arêtes responsable du conflit) que le MCCPS en cours de production ne contiendra pas. Lorsqu'un PCCS produit des conflits, nous stockons pour chaque conflit la correspondance d'arête (je stocke plutôt son indice dans **edgeMatchList**) qui en est responsable, et je diffère le traitement de ces conflits jusqu'à l'obtention du MCCPS.

Gestion des conflits : (Algorithme 5) Lorsqu'un PCCS donne un MCCPS, je gère tous les conflits qu'il a générés en créant pour chaque conflit une nouvelle branche d'extension contenant un nouveau PCCS à étendre. Pour obtenir le nouveau PCCS, je crée une copie du MCCPS en supprimant les nœuds et les arêtes en conflit avec la correspondance d'arêtes conflictuelle et en ajoutant la correspondance d'arêtes conflictuelle à la copie. La copie obtenue peut contenir plusieurs composants connexes, les autres composants connexes sont supprimés et seul le composant connexe contenant la correspondance d'arêtes conflictuelle est conservé. Deux cas sont possibles :

- Le composant connexe contient uniquement la correspondance d'arêtes responsable du conflit, dans ce cas, la nouvelle branche d'extension est détruite car elle donnera les mêmes MCCPS que l'extension avec la correspondance d'arêtes responsable du conflit comme point de départ.
- Le composant connexe ne contient pas seulement la correspondance d'arêtes responsable du conflit, dans ce cas, une nouvelle branche d'extension est créée avec le composant connexe.

Lors de l'extension à partir d'un point de départ s , tous les MCCPS trouvés doivent contenir s , donc les conflits impliquant des nœuds de s sont ignorés.

L'extension à partir d'un point de départ s , combinée aux conflits, permet de trouver tous les MCCPS contenant s . En effet, la gestion des conflits permet à l'algorithme d'extension de revenir sur ces choix et d'en faire d'autres pour trouver d'autres MCCPS. Cependant, un problème avec cette approche est que plusieurs conflits peuvent aboutir au même MCCPS. Pour éviter cela, avant de créer une nouvelle branche d'extension, je vérifie si le nouveau PCCS à étendre est inclus dans l'un des MCCPS déjà trouvés à partir du point de

départ s . Si c'est le cas, la branche n'est pas créée. Pour la vérification de l'inclusion, je recherche un isomorphisme de sous-graphe exact (qui préserve les étiquettes des arêtes, et l'identifiant des nœuds) (Algorithme 3).

Puisque l'extension à partir d'un point de départ s donne tous les MCCPS contenant s , alors l'extension à partir de différents points de départ peut donner des MCCPS identiques. Pour résoudre ce problème, lorsqu'une correspondance d'arêtes e peut être ajoutée à un PCCS résultant de l'extension à partir d'un point de départ s_i , alors je vérifie si $e \in \{s_0, s_1, s_2, \dots, s_{i-1}\}$, avec s_i , la correspondance d'arêtes à l'indice i dans **edgeMatchList**. Si c'est le cas, la correspondance d'arêtes n'est pas ajoutée au PCCS (Algorithme 4 ligne 10-12). Le sous-graphe obtenu à la fin de l'extension du PCCS peut ne pas être un MCCPS, c'est-à-dire qu'il peut ne pas être maximal. Si au moins l'une des correspondances d'arêtes e , qui ont été ignorées par le PCCS peut être ajoutée au sous-graphe obtenu alors il n'est pas maximal et n'est donc pas considéré comme un MCCPS (Algorithme 2 ligne 16-18).

Forme normale des sous-graphes : La forme normale que j'utilise pour représenter les sous-graphes est une liste d'adjacence. Lors de l'extension à partir d'un point de départ s , de nouvelles branches d'extension sont créées, chacune contenant un nouveau PCCS à étendre. Le souci est que le nombre de branches d'extension explose très rapidement, surtout pour de grandes instances de graphes. De plus, ces branches sont stockées en mémoire en attendant l'extension, ce qui consomme beaucoup de mémoire inutilement. Pour résoudre ce problème, j'ai développé une forme compressée pour représenter les sous-graphes.

Forme compressée des sous-graphes : Comme nous l'avons vu précédemment, les sous-graphes ne sont qu'un ensemble de correspondances d'arêtes, donc nous pouvons représenter un sous-graphe par un tableau d'entiers (forme compressée) contenant les indices dans **edgeMatchList** des correspondances d'arêtes qui le composent. La forme compressée utilise beaucoup moins de mémoire que la forme normale, avec un ratio d'environ 10. Pendant son cycle de vie, un sous-graphe est représenté sous cette forme compressée sauf pendant le processus d'extension. Le passage de la forme compressée à la forme normale se fait en $\theta(n)$, avec n le nombre d'arêtes du sous-graphe.

Algorithme 2: ExtensionOfEdgeMatches

Data: edgeMatchList, edgeMatchIdList1, edgeMatchIdList2

```
1 MCCPSList ← {} /* List of all MCCPS */
2 for i ← 0 to edgeMatchList.size() do
3   branches ← {};
4   s ← edgeMatchList[i] /* starting point */
5   branch ← createBranch();
6   branch.graphCompressedForm.append(i);
7   branches.append(branch);
8   MCCPSListTmp ← {} /* list of MCCPS from starting point s */
9   for branch in branches do
10    cf ← branch.graphCompressedForm;
11    if isIncluded(cf, MCCPSListTmp) ≠ 1 then
12      branch.graphNormalForm ← getNormalForm(cf);
13      result ← extendBranch(s, branch, edgeMatchList);
14      conflictingMatchesEdge ← result[0];
15      ignoredMatchesEdge ← result[1];
16      if isMaximal(branch.graphCompressedForm, ignoredMatchesEdge) then
17        MCCPSListTmp.append(branch.graphCompressedForm);
18        conflictManagement(edgeMatchList, conflictingMatchesEdge, branches,
19          MCCPSListTmp, s, branch);
19    free(branch);
20 MCCPSList.append(MCCPSListTmp);
```

La fonction `isIncluded` est utilisée pour le test d'isomorphisme exact de sous-graphe. En utilisant la forme compressée (ensemble d'entiers) d'un sous-graphe, mon algorithme a une complexité $\Theta(n * m)$, n et m étant le nombre d'arêtes des deux graphes comparés. La fonction `isIncluded` est également utilisée dans la fonction `conflictManagement`, car le nombre de MCCPS trouvés entre la création et l'extension d'une branche peut différer.

Algorithm 3: isIncluded

Data: graphCompressedForm, MCCPSList

```
1 for MCCPS in MCCPSList do
2   if graphCompressedForm.size() ≤ MCCPS.size() then
3     if graphCompressedForm ⊆ MCCPS then
4       return 1;
5 return 0;
```

Algorithm 4: extendBranch

Data: s, branch, edgeMatchList

```
1 edgeMatchListTmp ← {};
2 conflictingMatchesEdge ← {};
3 ignoredMatchesEdge ← {};
4 for node in branch.graphNormalForm.nodes() do
5   edgeMatchListTmp ← getEdgeMatchListContainNode(node) /* Returns edge matches that
6     contain the node as a parameter */
7   for edgeMatch in edgeMatchListTmp do
8     id ← edgeMatch.index;
9     if id not in branch.graphCompressedForm then
10      if canBeAdd(branch.graphNormalForm, edgeMatch) then
11        if s.index > id then
12          ignoredMatchesEdge.append(id);
13        else
14          branch.graphCompressedForm.append(id);
15          branch.graphNormalForm.append(edgeMatch);
16      else
17        if id not in conflictingMatchesEdge then
18          conflictingMatchesEdge.append(id);
```

Algorithme 5: conflictManagement

Data: edgeMatchList, conflictingMatchesEdge, branches, MCCPSList, s, branch

```
1 for index in conflictingMatchesEdge do
2   edgeMatch ← edgeMatchList[index];
3   nf ← branch.graphNormalForm;
4   nodes ← getConflictingNodes(nf);
5   subgraph ← copyGraphWhithoutNodes(nf,nodes);
6   subgraph.add(edgeMatch);
7   subgraph ← removeComponent(subgraph, edgeMatch)/* Remove the other connected
   components and keep the connected component containing edgeMatch */
8   if subgraph.size() > 1 and s in subgraph then
9     if isIncluded(subgraph, MCCPSList) ≠ 1 then
10      newBranch ← createBranch();
11      newBranch.graphCompressedForm ← getCompressedForm(subgraph);
12      free(subgraph);
13      branches.append(newBranch);
```

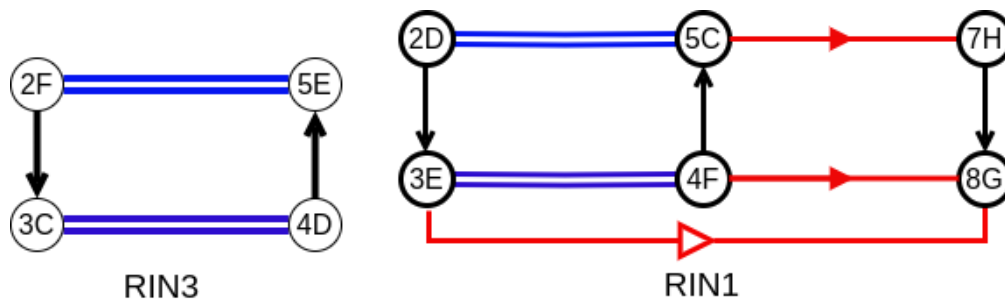


Figure 2.1 En considérant les graphes G et H de la figure 1.6, j’obtiens également RIN3, qui est inclus dans RIN1 et contient les mêmes arêtes que RIN1 dans G et H (c’est-à-dire, ils cartographient les mêmes régions de G et H). RIN3 sera supprimé car il représente une fausse variation de RIN1.

2.1.2.1 Élimination des MCCPS partageant le même ensemble d’arêtes

J’élimine les sous-graphes qui ont les mêmes ensembles d’arêtes dans g et h, car les motifs structuraux récurrents doivent se trouver dans des emplacements non homologues. En effectuant une comparaison par paire des MCCPS trouvés, deux cas sont possibles :

- **Isomorphisme de sous-graphe** : Le plus petit MCCPS (en termes de nombre d’arêtes) est éliminé.

— **Isomorphisme de graphe** : Un des 2 MCCPS est éliminé.

Pour le test d'isomorphisme, je fais la même chose que dans la fonction `isIncluded`, sauf que j'utilise les identifiants dans `edgeMatchIdList1` et `edgeMatchIdList2`.

2.2 Parallélisation de l'algorithme pour les Sous-graphes Partiels Communs Connexes Maximaux (MCCPS)

Les machines actuelles présentes dans les systèmes hautes performances sont constituées d'une agrégation de machines multicœurs, parfois équipées d'accélérateurs (GPU, FPGA, ...) et interconnectées par un réseau à haute vitesse. L'organisation de la mémoire sur ces systèmes est telle qu'ils peuvent être programmés en utilisant une combinaison de deux modèles de mémoire. Les éléments de traitement fonctionnant sur le même nœud partagent la mémoire orchestrée par un seul système d'exploitation, ce qui permet de tirer parti du *modèle de mémoire partagée*. Les éléments de traitement fonctionnant sur différents nœuds ne partagent physiquement aucune mémoire, ce qui rend nécessaire la distribution des données sur plusieurs espaces mémoire, orchestrée par autant d'instances de systèmes d'exploitation, en utilisant le *modèle de mémoire distribuée*. Il convient de noter cependant que plusieurs éléments fonctionnant sur une même machine peuvent utiliser le modèle de mémoire distribuée, et que certains modèles de programmation distribués masquent l'organisation distribuée de la mémoire sous une couche d'abstraction pour la faire ressembler à un seul espace mémoire, tel que Unified Parallel C.

La combinaison de ces deux modèles conduit à une *parallélisation hiérarchique* suivant un *modèle de mémoire hybride*, où les éléments de traitement fonctionnant sur le même nœud coopèrent ensemble en utilisant la mémoire partagée, et les ensembles d'éléments de traitement sur plusieurs nœuds suivent le modèle de mémoire distribuée pour travailler ensemble. Je présente l'algorithme de recherche qui s'exécute sur un seul nœud dans la section sur mémoire partagée 2.2.1, et je présente l'algorithme sur mémoire distribuée dans la section 2.2.2.

2.2.1 Algorithme sur mémoire partagée

Je me suis basé sur l'approche **PCAM**, qui comprend les étapes suivantes :

- **Partitionnement** : Le problème est décomposé en tâches de granularité très fine.
- **Communication** : Identification des dépendances entre les tâches.
- **Agglomération** : Une fois que les tâches et leurs dépendances ont été déterminées dans les étapes précédentes, les tâches peuvent, si nécessaire, être combinées en tâches plus grandes pour améliorer les performances (réduire les coûts de communication).
- **Mapping** : Répartition des tâches sur les unités d'exécution.

Je définis ici des mots clé utilisés dans les algorithmes parallèles qui seront présenter dans la suite du document.

- **Parallel construction** et **End parallel construction** sont utilisés pour délimiter une région parallèle. Les instructions comprises dans une région parallèle sont destinées à être exécutées simultanément par les unités d'exécution.
La portée de chaque objet créé dans une région **Parallel construction** est locale à la région parallèle, et les objets sont privés à chaque thread.
- **shared** est utilisé pour partager des données entre les unités d'exécution d'une région parallèle.
- **parallel for** est utilisé pour exécuter une boucle en parallèle. Les itérations de la boucle peuvent être exécutées simultanément par les unités d'exécution.
- **begin critical section** et **end critical section** sont utilisés pour définir une section critique dans une région parallèle. Les instructions situées dans une section critique seront exécuter séquentiellement par les unités d'exécution, une à la fois, et non simultanément. Une section critique est utilisée pour garantir l'intégrité des données partagées entre les unités d'exécution. En effet, lorsque plusieurs unités d'exécution tentent d'accéder ou de modifier une même donnée en même temps, cela peut entraîner des problèmes de cohérence et de concurrence.

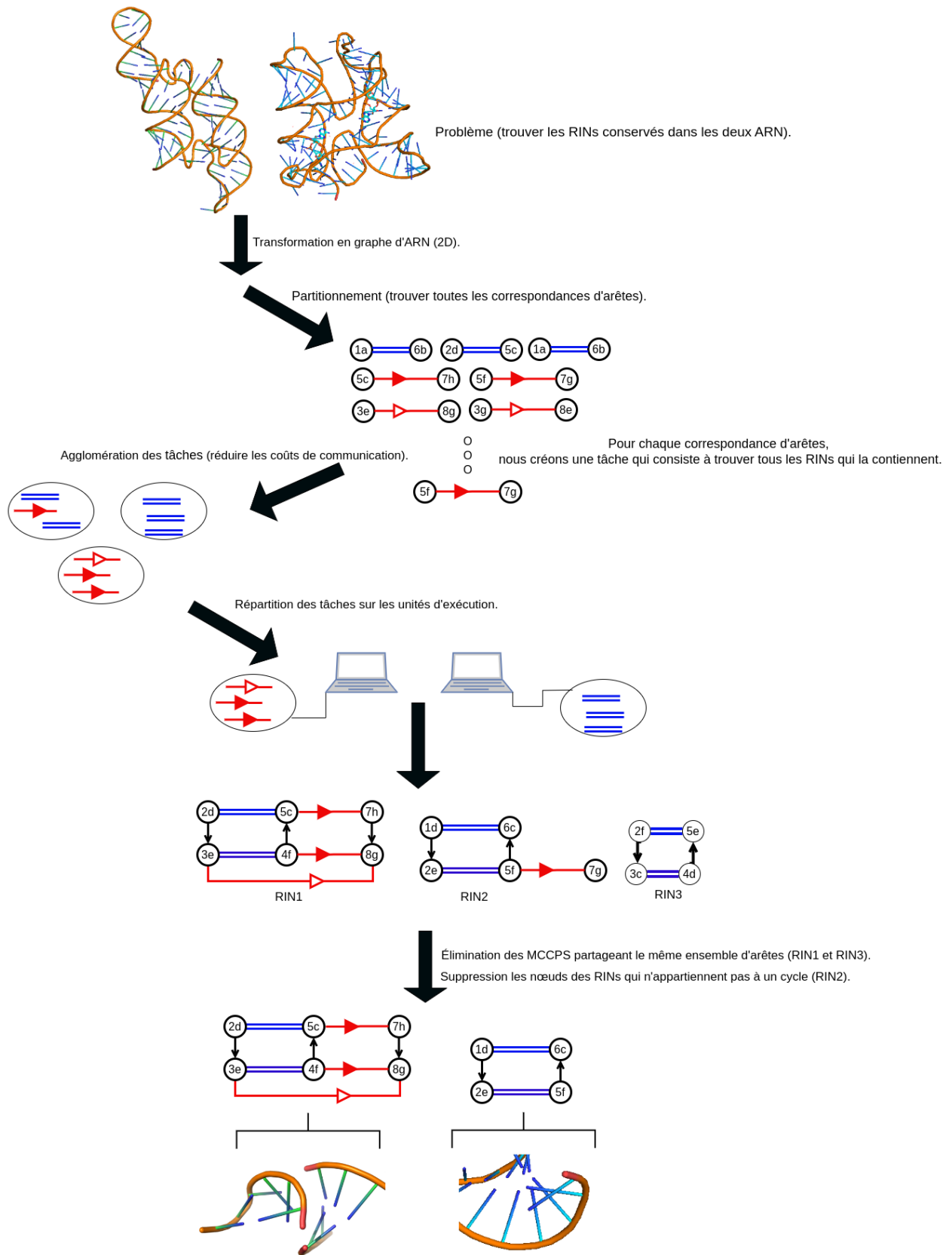


Figure 2.2 Pipeline de recherche des RINs basé sur l'approche PCAM.

2.2.1.1 Recherche de toutes les correspondances d'arêtes

Pour l'étape de partitionnement, j'ai effectué une décomposition de domaine unidimensionnelle sur edgeTypeList afin d'avoir une tâche pour chaque élément de edgeTypeList. J'ai obtenu des tâches indépendantes qui sont associées à des threads en utilisant une répartition dynamique.

Algorithme 6: SearchAllEdgeMatches Parallel

Data: g, h, edgeTypeList

```
1 edgeMatchList ← {} /* shared variable */
2 edgeMatchIdList1 ← {} /* shared variable */
3 edgeMatchIdList2 ← {} /* shared variable */
4 Parallel construction shared(g, h, edgeTypeList);
5 edgeMatchListLocal ← {};
6 edgeMatchIdList1Local ← {};
7 edgeMatchIdList2Local ← {};
8 parallel for  $i \leftarrow 0$  to edgeTypeList.size() do
9     edgeType ← edgeTypeList[i];
10    label ← edgeType.edgeLabel;
11    edgeList1 ← getOutgoingEdge(g,label);
12    edgeList2 ← getOutgoingEdge(h,label);
13    for edge1 in edgeList1 do
14        for edge2 in edgeList2 do
15            matchEdge ← createMatchEdge(edge1,edge2);
16            edgeMatchListLocal.append(matchEdge);
17            edgeMatchIdList1Local.append(edge1.id);
18            edgeMatchIdList2Local.append(edge2.id);
19 begin critical section;
20    edgeMatchList.append(minimumSubGraphListLocal);
21    edgeMatchIdList1.append(minimumSubGraphIdList1Local);
22    edgeMatchIdList2.append(minimumSubGraphIdList2Local);
23 end critical section;
24 End parallel construction;
```

2.2.1.2 Extension des correspondances d'arêtes

J'ai effectué une décomposition de domaine unidimensionnelle sur l'ensemble des points de départ pour avoir une tâche pour chaque point de départ. J'ai utilisé des tâches de granularité fine car la charge de travail est très déséquilibrée pour les extensions à partir d'un point de départ. Pour la distribution des tâches sur les unités d'exécution, j'ai eu recours à différentes stratégies d'ordonnancement pour essayer d'équilibrer la charge de travail.

2.2.1.2.1 Version avec distribution dynamique

Dans la version avec distribution dynamique, j'ai utilisé une distribution dynamique des tâches, un thread reçoit une tâche correspondant à l'extension à partir d'un point de départ et dès qu'il a terminé, il passe à une autre. Pour cette approche, il n'est pas nécessaire de communiquer ou de synchroniser entre les tâches car elles sont indépendantes. Cependant, cette approche souffre d'un déséquilibre de charge important lorsque la distribution du nombre de MCCPS par point de départ est très déséquilibrée.

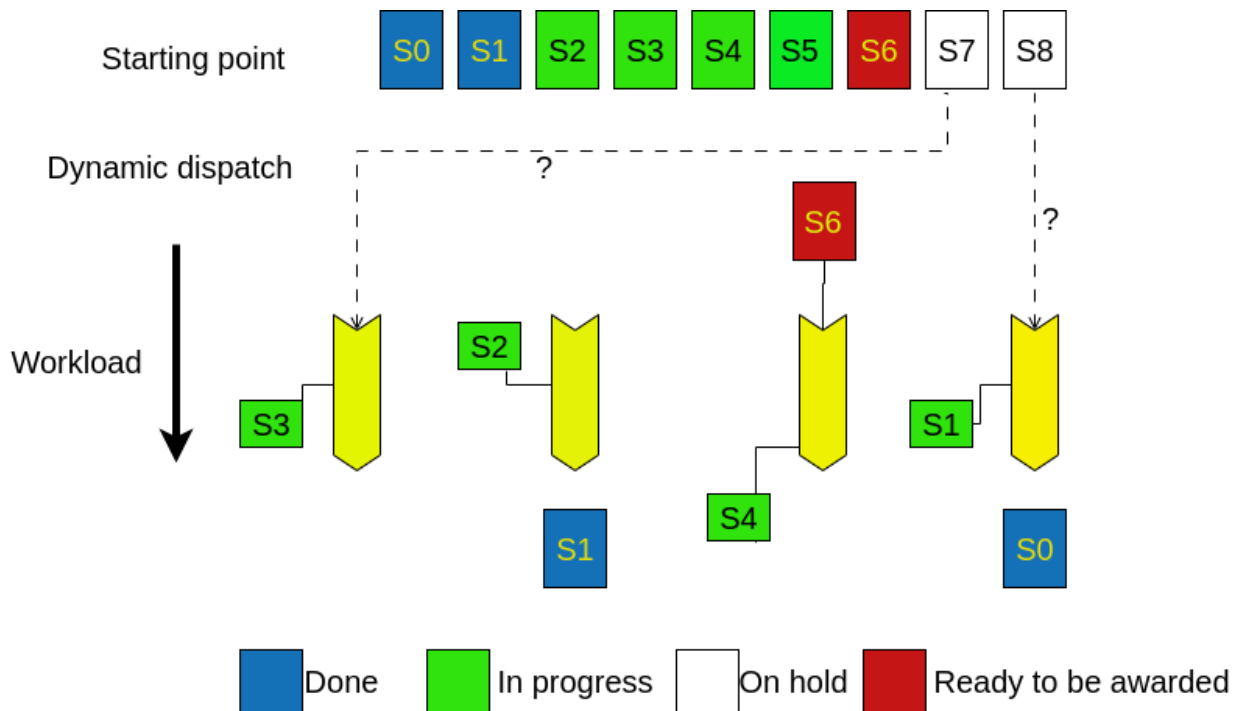


Figure 2.3 Distribution dynamique des tâches : les points de départ sont divisés en paquets de taille 1. Dès qu'un thread (en jaune) termine l'extension de son point de départ, un autre paquet lui est attribué.

Algorithm 7: ExtensionOfEdgeMatches Parallel

Data: edgeMatchList, edgeMatchIdList1, edgeMatchIdList2

```
1 MCCPSList ← {} /* shared variable */
2 Parallel construction shared(edgeMatchList, edgeMatchIdList1, edgeMatchIdList2);
3 MCCPSListLocal ← {};
4 parallel for  $i \leftarrow 0$  to edgeMatchList.size() do
5   branches ← {};
6    $s \leftarrow$  edgeMatchList[i] /* starting point */
7   branch ← createBranch();
8   branch.graphCompressedForm.append(i);
9   branches.append(branch);
10  MCCPSListTmp ← {} /* list of MCCPS from starting point s */
11  for branch in branches do
12    cf ← branch.graphCompressedForm;
13    if isIncluded(cf, MCCPSListTmp)  $\neq 1$  then
14      branch.graphNormalForm ← getNormalForm(cf);
15      result ← extendBranch(s, branch, edgeMatchList);
16      conflictingMatchesEdge ← result[0];
17      ignoredMatchesEdge ← result[1];
18      if isMaximal(branch.graphCompressedForm, ignoredMatchesEdge) then
19        MCCPSListTmp.append(branch.graphCompressedForm);
20        conflictManagement(edgeMatchList, conflictingMatchesEdge, branches,
21          MCCPSListTmp, s, branch);
21    free(branch);
22  MCCPSListLocal.append(MCCPSListTmp);
23 begin critical section;
24  MCCPSList.append(MCCPSListLocal);
25 end critical section;
26 sameEdgePostProcessing(MCCPSList) /* Parallel */
27 End parallel construction;
```

2.2.1.2.2 Version imbriquée

Pour la version imbriquée, j'utilise des groupes de threads, un groupe reçoit une tâche d'extension à partir d'un point de départ et dès qu'il a terminé, il passe à un autre. Le thread maître de chaque groupe reçoit une tâche d'extension à partir d'un point de départ et les tâches d'extension de branche résultant de la gestion des conflits sont distribuées parmi les threads du même groupe. Les threads du même groupe doivent se synchroniser pour éviter d'étendre une branche qui donnera un MCCPS déjà trouvé.

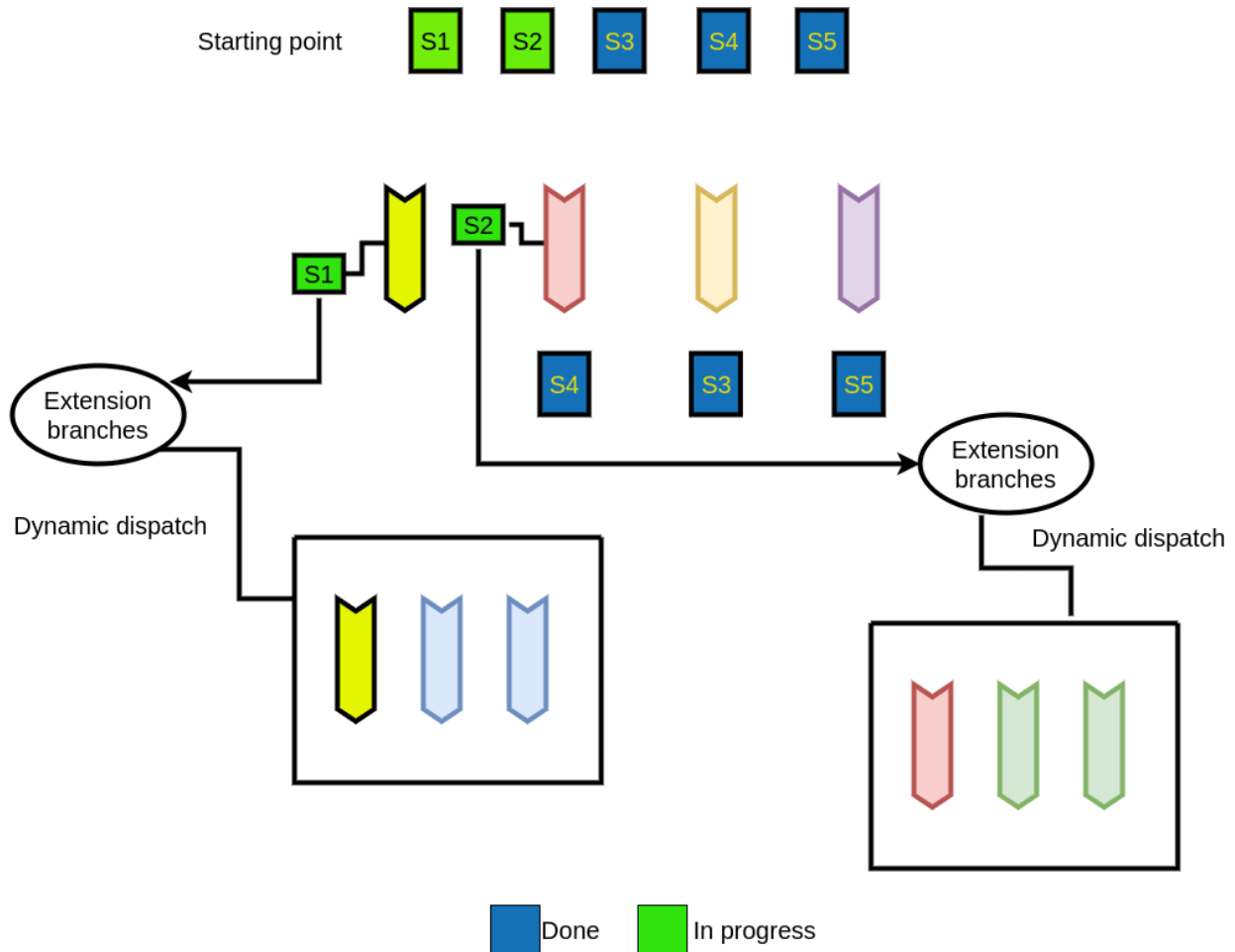


Figure 2.4 Répartition des tâches pour la version imbriquée : lorsqu'un point d'extension est attribué à un thread, celui-ci crée d'autres threads pour partager les tâches d'extension de branche.

Algorithm 8: ExtensionOfEdgeMatches Parallel nested version

Data: edgeMatchList, edgeMatchIdList1, edgeMa1.2chldList2

```
1 MCCPSList ← {} /* shared variable */
2 Parallel construction shared(edgeMatchList, edgeMatchIdList1, edgeMatchIdList2);
3 MCCPSListLocal ← {};
4 parallel for  $i \leftarrow 0$  to edgeMatchList.size() do
5   branches ← {};
6    $s \leftarrow$  edgeMatchList[i] /* starting point */
7   branch ← createBranch();
8   branch.graphCompressedForm.append(i);
9   branches.append(branch);
10  MCCPSListTmp ← {} /* list of MCCPS from starting point s */
11  Parallel construction private(branch);
12  branchesSize ← 0;
13  branchesSizeTmp ← 0;
14  while branchesSizeTmp < branches.size() do
15    branchesSize ← branches.size();
16    parallel for  $j = \text{branchesSizeTmp}$  to branchesSize do
17      branch ← branches[j];
18      cf ← branch.graphCompressedForm;
19      if isIncluded(cf, MCCPSListTmp)  $\neq 1$  then
20        branch.graphNormalForm ← getNormalForm(cf);
21        result ← extendBranche(s, branch, edgeMatchList);
22        conflictingMatchesEdge ← result[0];
23        ignoredMatchesEdge ← result[1];
24        if isMaximal(branch.graphCompressedForm, ignoredMatchesEdge) then
25          begin critical section;
26          MCCPSListTmp.append(branch.graphCompressedForm);
27          conflictManagement(edgeMatchList, conflictingMatchesEdge, branches, MCCPSListTmp, s,
28            branch);
29          end critical section;
30        free(branch);
31      branchesSizeTmp ← branchesSize;
32  End parallel construction;
33  MCCPSListLocal.append(MCCPSListTmp);
34 begin critical section;
35  MCCPSList.append(MCCPSListLocal);
36 end critical section;
37 sameEdgePostProcessing(MCCPSList) /* Parallel */
End parallel construction;
```

2.2.1.2.3 Version avec vol de travail

Dans la version avec vol de travail, nous avons le même fonctionnement que dans la version avec distribution dynamique, sauf que lorsque les tâches d'extension d'un point de départ sont épuisées, les threads inactifs viennent voler les tâches d'extension de branche des threads actifs. Cette approche permet un meilleur équilibrage de la charge de travail.

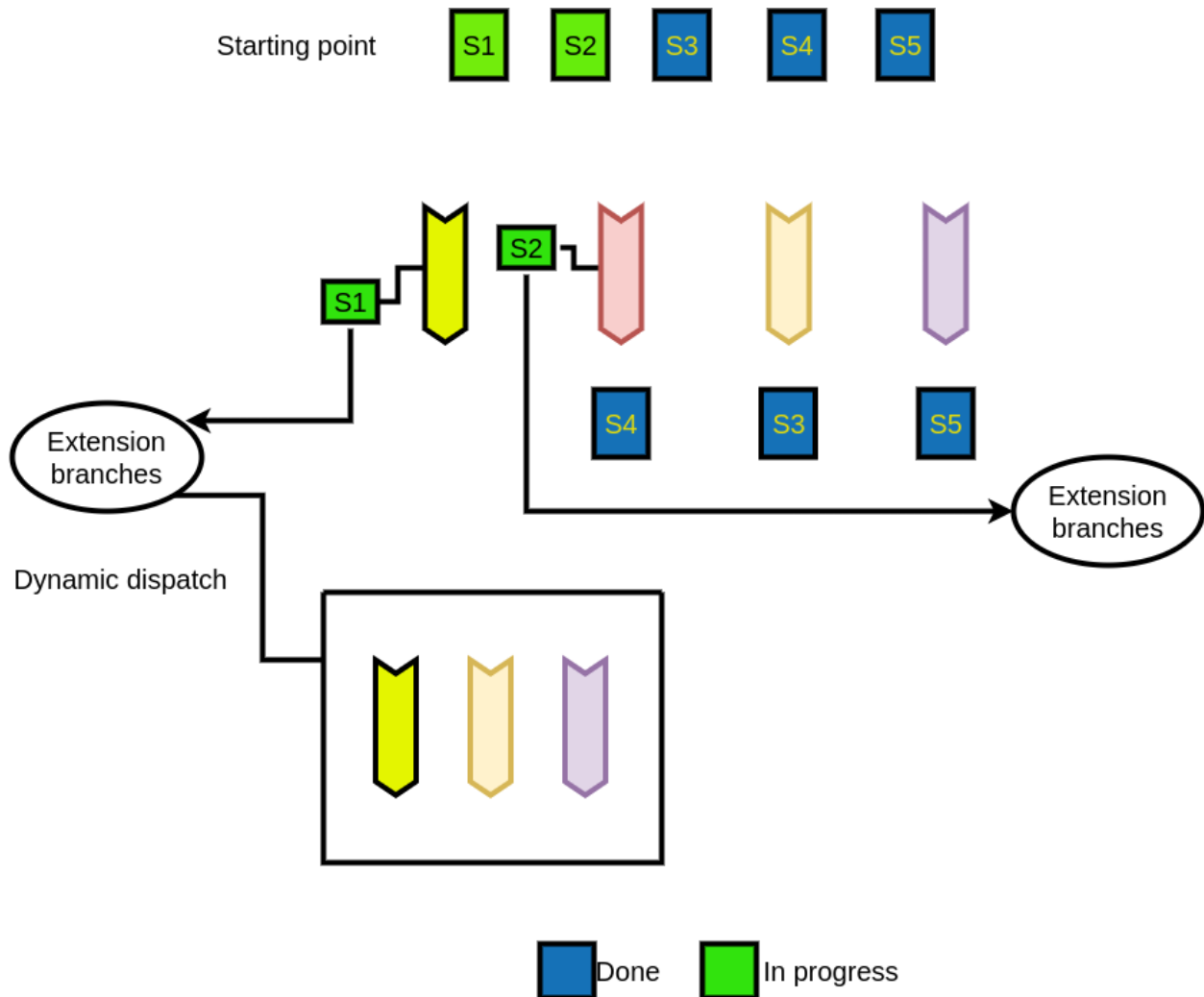


Figure 2.5 Répartition des tâches pour la version avec vol de travail : lorsqu'un thread n'a plus de tâches à effectuer, il vole les tâches d'extension de branche d'un autre thread.

Afin de réduire le coût de synchronisation des tâches et d'éviter le partage inutile de tâches, une stratégie est mise en place où les threads ne partagent pas directement leurs tâches d'extension de branche avec les autres threads. Au lieu de cela, les threads conservent leurs tâches jusqu'à ce qu'un besoin de partage se

manifeste. Ce besoin est détecté lorsqu'il est confirmé qu'au moins un autre thread est inactif ou disponible pour recevoir de nouvelles tâches (Algorithme 9 ligne 19-22). Ce mécanisme permet d'améliorer l'efficacité en évitant les partages inutiles, tout en garantissant que les threads disponibles peuvent être utilisés de manière optimale. Un thread sait s'il y a au moins un autre thread disponible grâce à la variable partagée **numThreadFree**. Lorsqu'il n'y a plus de tâches d'extension à partir d'un point de départ, les threads libres définissent la valeur à l'indice **rankThread** dans **threadStateList** à 1, indiquant qu'ils sont libres, et incrémentent **numThreadFree** de 1 (Algorithme 9 ligne 44-46).

Pour continuer ainsi, lorsqu'au moins un thread est disponible, seul l'un des threads occupés partage ses tâches d'extension de branche, et c'est celui avec le rang le plus petit (**isYourRound**). La fonction **isYourRound** prend **threadStateList** et **rankThread** en entrée. Elle parcourt **threadStateList** depuis le début jusqu'à l'index **rankThread-1**, et si elle trouve seulement des 1, elle retourne 1, indiquant que le thread appelant la fonction devrait partager ses tâches. Sinon, la fonction retourne 0.

Algorithm 9: ExtensionOfEdgeMatches Parallel work stealing version

Data: edgeMatchList, edgeMatchIdList1, edgeMatchIdList2

```
1 MCCPSList ← {} /* shared variable */
2 threadStateList ← {} /* shared variable */
3 numThreadFree ← 0 /* shared variable */
4 Parallel construction shared(edgeMatchList, edgeMatchIdList1, edgeMatchIdList2);
5 MCCPSListLocal ← {};
6 shareTask ← 0;
7 parallel for  $i \leftarrow 0$  to  $edgeMatchList.size()$  do
8   branches ← {};
9    $s \leftarrow edgeMatchList[i]$  /* starting point */
10  branch ← createBranch();
11  branch.graphCompressedForm.append(i);
12  branches.append(branch);
13  MCCPSListTmp ← {} /* list of MCCPS from starting point s */
14  branchesSize ← 0;
15  branchesSizeTmp ← 0;
16  while  $branchesSizeTmp < branches.size()$  do
17    branchesSize ← branches.size();
18    for  $j=branchesSizeTmp$  to  $branchesSize$  do
19      if not shareTask and numThreadFree > 0 then
20        if isYourRound(threadStateList, threadRank) then
21          shareTask ← 0;
22        Do Task if(shareTask) private(branch);
23        branch ← branches[j];
24        cf ← branch.graphCompressedForm;
25        if isIncluded(cf, MCCPSListTmp)  $\neq 1$  then
26          branch.graphNormalForm ← getNormalForm(cf);
27          result ← extendBranch(s, branch, edgeMatchList);
28          conflictingMatchesEdges ← result[0];
29          ignoredMatchesEdge ← result[1];
30          if isMaximal(branch.graphCompressedForm, ignoredMatchesEdge) then
31            begin critical section;
32            MCCPSListTmp.append(branch.graphCompressedForm);
33            conflictManagement(edgeMatchList, conflictingMatchesEdge, branches, MCCPSListTmp, s,
34              branch);
35            end critical section;
36          free(branch);
37          End Task;
38        if shareTask then
39          Task Wait;
40        branchesSizeTmp ← branchesSize;
41      MCCPSListLocal.append(MCCPSListTmp);
42  begin critical section;
43  MCCPSList.append(MCCPSListLocal);
44  end critical section;
45  threadStateList[threadRank] ← 1;
46  begin critical section;
47  numThreadFree++;
48  end critical section;
49  sameEdgePostProcessing(MCCPSList) /* Parallel */
End parallel construction;
```

2.2.2 Algorithme sur mémoire distribuée

2.2.2.1 Recherche de toutes les correspondances d'arêtes

Chaque processus exécute le même algorithme pour la mémoire partagée (Algorithme 6). Cela permet d'éliminer le coût de communication entre les processus.

2.2.2.2 Extension des correspondances d'arêtes

Un lot de points de départ est envoyé aux processus, chacun utilisant la version de vol de travail de l'algorithme 9 pour l'extension. Pour la distribution des paquets entre les processus, j'ai utilisé le schéma Maître/Travailleur.

J'aurai pu implémenter la version de vol de travail pour le modèle de mémoire distribuée, mais la synchronisation entre les tâches d'extension de branche aurait entraîné une quantité importante de communication entre les processus, rendant cette approche inefficace.

2.3 Des MCCPS aux RINs

Ici, je présente l'ensemble de contraintes associées aux RINs que j'ai utilisé pour réduire l'espace de recherche de l'algorithme de recherche des MCCPS. Soit r un RIN :

1. Les RINs sont composés d'interactions non-canoniques, ainsi l'ensemble des points de départ a été réduit à l'ensemble des correspondances d'arêtes correspondants aux interactions non-canoniques.
2. Si deux nœuds, n_1 et n_2 , dans r forment une paire de bases canonique locale, il existe un nœud n_3 dans r tel que n_3 est un voisin de n_1 ou n_2 , et n_3 est impliqué dans une interaction non-canonique. En d'autres termes, nous n'étendons pas les empilements dont les nucléotides sont impliqués uniquement que dans des paires de bases canoniques.
3. Chaque nœud appartient à un cycle dans le graphe non dirigé induit par un RIN, cette contrainte est satisfaite en post-traitement en supprimant les nœuds des RINs qui n'appartiennent pas à un cycle.

2.4 Recherche des MCCPS contenant un motif donné

Un algorithme naïf pour trouver des MCCPS contenant un motif donné serait de trouver tous les MCCPS et d'utiliser un algorithme d'isomorphisme de sous-graphe pour filtrer les MCCPS qui ne contiennent pas le motif. Je présente un algorithme plus efficace que le naïf, dans le sens où il permet de réduire l'espace de

recherche en se concentrant uniquement sur les MCCPS contenant le motif recherché, et évite ainsi l'étape de filtrage des MCCPS.

L'algorithme commence par rechercher des sous-isomorphismes du motif dans les deux graphes d'entrée. Ensuite, nous associons chaque sous-isomorphisme dans le premier graphe d'entrée avec ceux dans le second, en utilisant l'identifiant des nœuds du motif (un nœud dans le premier graphe d'entrée est associé à un nœud dans le second s'ils correspondent au même nœud dans le motif). Les correspondances obtenues sont des PCCS, et sont ensuite étendues pour trouver tous les MCCPS contenant le motif.

Ce nouvel algorithme présente l'avantage de réduire considérablement l'espace de recherche, en ne considérant que les MCCPS potentiellement pertinents, ce qui peut améliorer de manière significative les performances, surtout pour les grands ensembles de données. Je désignerai désormais la recherche de MCCPS contenant un motif donné comme **l'extension d'un motif**.

Dans le chapitre 3 nous allons comparer et évaluer les différentes versions parallèles de l'algorithme pour identifier les MCCPS.

CHAPITRE 3

RÉSULTATS

Tout d'abord, je présenterai l'évaluation des algorithmes parallèles afin de choisir le meilleur, puis je décrirai les RINs découverts.

Métriques : Les algorithmes ont été évalués en termes de leur évolutivité (scalability). L'évolutivité est une mesure importante de la performance d'un algorithme parallèle. L'évolutivité permet d'évaluer le comportement de l'algorithme lorsque le nombre d'unités de traitement (cœurs, threads, etc.) augmente. Une bonne évolutivité signifie que l'algorithme peut pleinement utiliser les ressources supplémentaires lorsqu'il est exécuté en parallèle, et que le temps d'exécution diminue proportionnellement au nombre d'unités de traitement.

Environnement expérimental : Les expériences ont été réalisées sur le cluster Narval¹. Chaque nœud Narval possède soit 48 cœurs, soit 64 cœurs et est équipé de 2 processeurs AMD MILAN ou ROME (7502, 7532, 7413), fonctionnant à des fréquences comprises entre 2,40 et 2,65 GHz, avec des caches L3 de 128 Mo ou 256 Mo. La RAM disponible sur chaque nœud varie entre 498 Go et 4000 Go.

Implémentation des algorithmes : Les algorithmes ont été implémentés en langage C au sein d'un solveur nommé PASlgraph². J'ai utilisé OpenMP (mémoire partagée) et MPI (mémoire distribuée) pour la parallélisation.

Le solveur PASlgraph prend en entrée deux graphes orientés et retourne tous les MCCPS des deux graphes. De plus, il offre la possibilité de trouver tous les MCCPS contenant un motif donné. PASlgraph propose plusieurs options pour paramétrer son utilisation. Parmi ces options, on trouve la possibilité de définir la liste de description des types d'arêtes (voir 2.1.1), ainsi que des options pour relâcher ou désactiver les contraintes sur les RINs définies dans la section 2.3.

Par défaut, seules les correspondances d'arêtes correspondants aux interactions non-canoniques constituent l'ensemble des points de départ. Cependant, il existe une option pour spécifier un fichier contenant les types d'arêtes qui vont constituer l'ensemble des points de départ. De plus, des options sont disponibles pour désactiver les contraintes 2 et 3 sur les RINs, qui sont activées par défaut. Le nombre de threads utili-

1. <https://docs.alliancecan.ca/wiki/Narval>

2. https://gitlab.info.uqam.ca/agbeto.kossi_wilfried/isomorphisme_graphe

sés par le solveur peut également être défini selon les besoins de l'utilisateur. Ces paramètres offrent une grande flexibilité dans l'utilisation du solveur PASgraph et permettent de l'adapter aux besoins spécifiques de chaque utilisateur.

Les expériences ont été réalisées en utilisant les paramètres par défaut du solveur.

Jeux de données : J'ai appliqué ma méthode à toutes les paires de structures d'ARN non-redondantes pour trouver tous les modules structurels. Les données proviennent de la base de données d'ARN non-redondantes maintenue sur RNA3DHub. J'ai utilisé la version 3.269, tandis que les projets (Reinharz *et al.*, 2018) et (Soulé *et al.*, 2021) utilisent respectivement les versions 2.92 et 3.137.

Le jeu de données est divisé en 2 classes (Tableau 3.1) :

- Globale : Contient des structures d'ARN complètes.
- Locale : Contient des boucles multiples, des boucles intérieures, des renflements et des épingles à cheveux.

Table 3.1 Pour chaque classe, je donne le nombre d'instances (#inst) et ensuite je décris les caractéristiques du graphe : nombre minimum et maximum de nœuds, nombre d'arêtes minimum et maximum.

Classe	#Inst	Nœuds (min)	Nœuds (max)	Arêtes (min)	Arêtes (max)
Global	1722	2	16770	2	37582
Local	16431	2	5066	2	10132

La classe globale contient 200 graphes avec un nombre de nœuds supérieur ou égal à 100, tandis que la classe locale en contient 75.

3.1 Performance de la parallélisation

Tout d'abord, j'ai évalué les performances de l'implémentation sur mémoire partagée sur un seul nœud. La figure 3.1 présente l'évolutivité des 3 versions de l'algorithme en mémoire partagée sur 2 graphes sélectionnés aléatoirement dans notre ensemble de données. La figure 3.2 montre leurs temps d'exécution sur 7 comparaisons de graphes différentes sur 10 cœurs. Les résultats montrent que la version avec vol de travail performe mieux que les autres. Cette meilleure performance peut s'expliquer par le fait que la version avec vol de travail équilibre mieux la charge de travail, ce qui est confirmé par les mesures d'efficacité du CPU (Version avec vol de travail : 94%, version imbriquée : 75%, version dynamique : 57%).

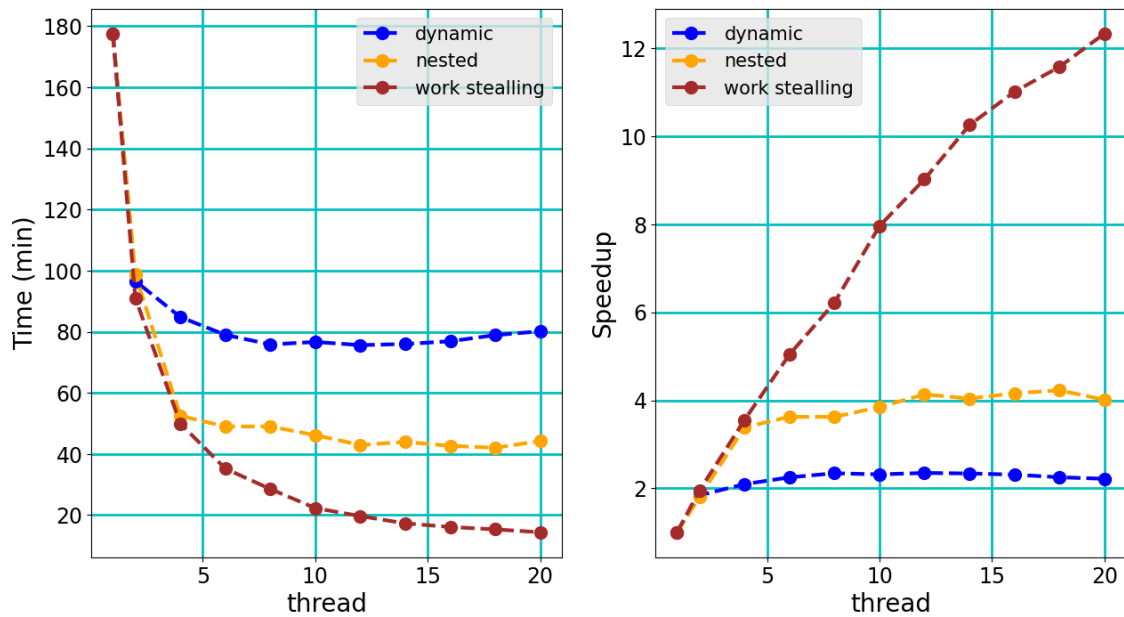


Figure 3.1 L'évaluation de l'évolutivité pour toutes les versions avec OpenMP. Les deux graphes comparés ont respectivement 390 et 1800 nœuds, et 1003 et 4282 arêtes.

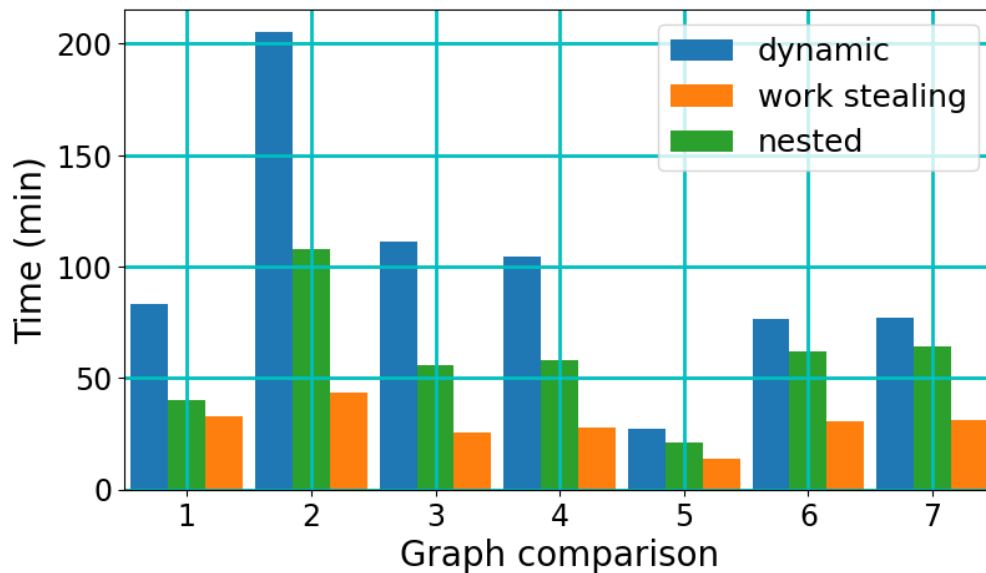


Figure 3.2 Le temps d'exécution de 7 comparaisons de graphes sur 10 threads pour toutes les versions avec OpenMP.

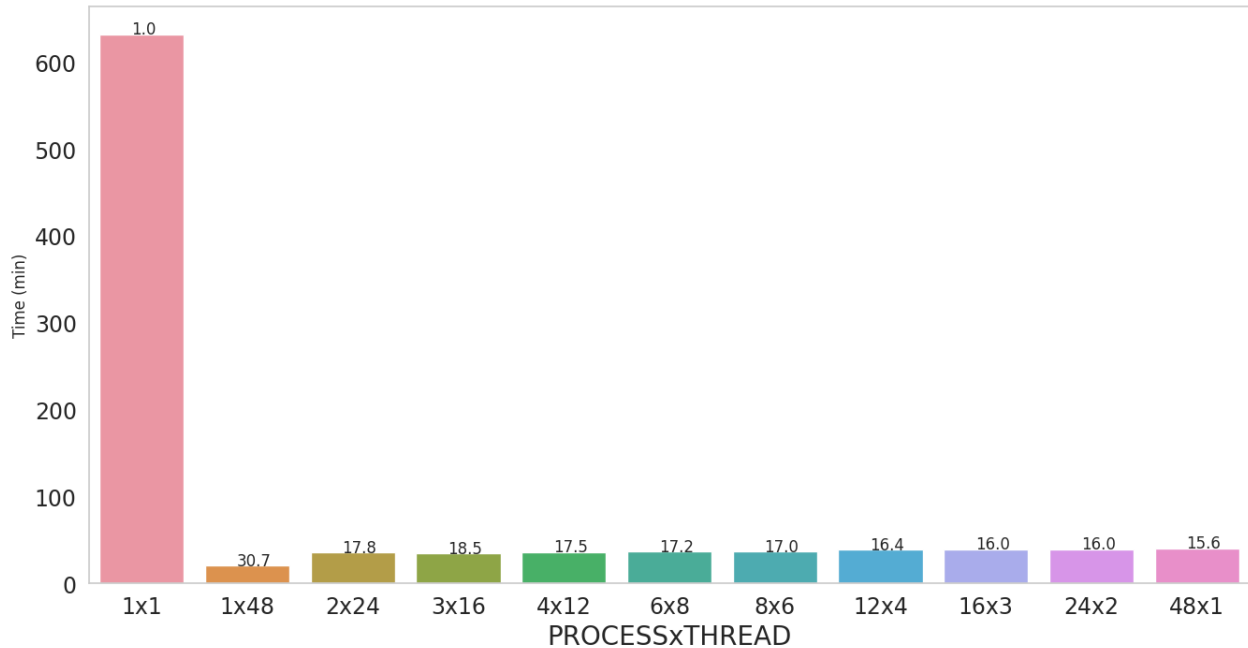


Figure 3.3 Comparaison des performances (en termes de temps d'exécution et d'accélération) de la version hybride pour différentes allocations $P \times T = 48$, où P est le nombre de processus et T est le nombre de threads. Le numéro sur chaque barre de l'histogramme représente l'accélération.

La version avec vol de travail est donc utilisée dans les autres expériences présentées dans le document.

Pour exploiter au mieux l'architecture hiérarchique des machines, je peux utiliser une combinaison de l'algorithme en mémoire partagée et d'un algorithme en mémoire distribuée. L'algorithme en mémoire partagée est utilisé au sein des nœuds de calcul, et les instances de l'algorithme en mémoire partagée sont regroupées par l'algorithme en mémoire distribuée, ce qui permet le calcul sur plusieurs nœuds. Le défi ici est de trouver la bonne répartition entre le nombre de threads (utilisés par l'algorithme en mémoire partagée) par processus (utilisé par l'algorithme en mémoire distribuée). Pour un nombre donné de cœurs (48), j'ai comparé les performances de diverses allocations $P \times T = 48$, où P est le nombre de processus et T est le nombre de threads. Les résultats sont présentés dans la figure 3.3. On peut observer que le temps d'exécution le plus bas est atteint avec un seul processus et le nombre maximal de threads. Par conséquent, j'utilise cette allocation dans les autres expériences présentées dans cette section.

La Figure 3.4 présente l'évolutivité de l'algorithme hybride (mémoire partagée et distribuée). J'ai utilisé 4 à 10 threads par processus, un processus par nœud, allant de 1 à 12 nœuds. Pour un nombre donné d'unités d'exécution, l'algorithme en mémoire partagée est plus rapide que l'algorithme en mémoire distribuée,

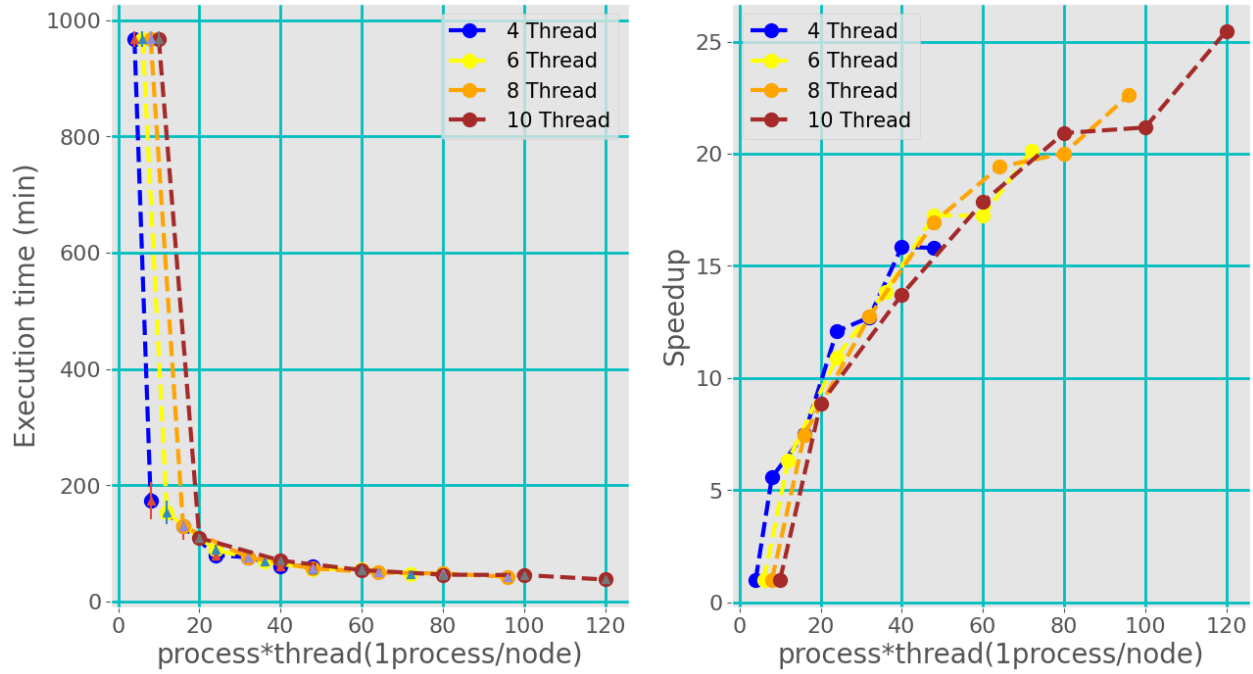


Figure 3.4 Évaluation de l'évolutivité de la version hybride. Les deux graphes comparés ont respectivement 1447 et 1949 nœuds, et 3492 et 4412 arêtes. 1 seul processus est utilisé par nœud.

car en cas de déséquilibre de charge, les processus inactifs ne peuvent pas obtenir de travail des autres processus, alors que l'algorithme en mémoire partagée dispose d'un mécanisme de vol de travail.

3.2 Description des RINs trouvés

J'ai utilisé PASigraph pour effectuer une comparaison par paire des structures d'ARN sur les ensembles de données global et local afin d'identifier tous les RINs qu'ils contiennent. Cela a pris 3 semaines pour l'ensemble de données local et 4 semaines pour l'ensemble de données global. Ce qui est significativement plus rapide que dans les travaux précédents (Reinharz *et al.*, 2018; Soulé *et al.*, 2021). Cela a nécessité un total de 186 Go de mémoire. Il est important de noter que les résultats obtenus n'étaient pas réalisables avec les méthodes proposées dans les travaux précédents, même avec 4 To de RAM. De plus, j'ai trouvé un nombre plus important de RINs.

3.2.1 RINs Locaux

L'ensemble de données local se compose de 16 431 structures d'ARN représentant des boucles multiples, des boucles intérieures, des renflements et des épingles à cheveux. J'ai trouvé 631 RINs pour un total de 182 646 occurrences. Le RIN le plus petit comporte 3 nœuds et 6 arêtes, tandis que le plus grand comporte 19 nœuds et 86 arêtes.

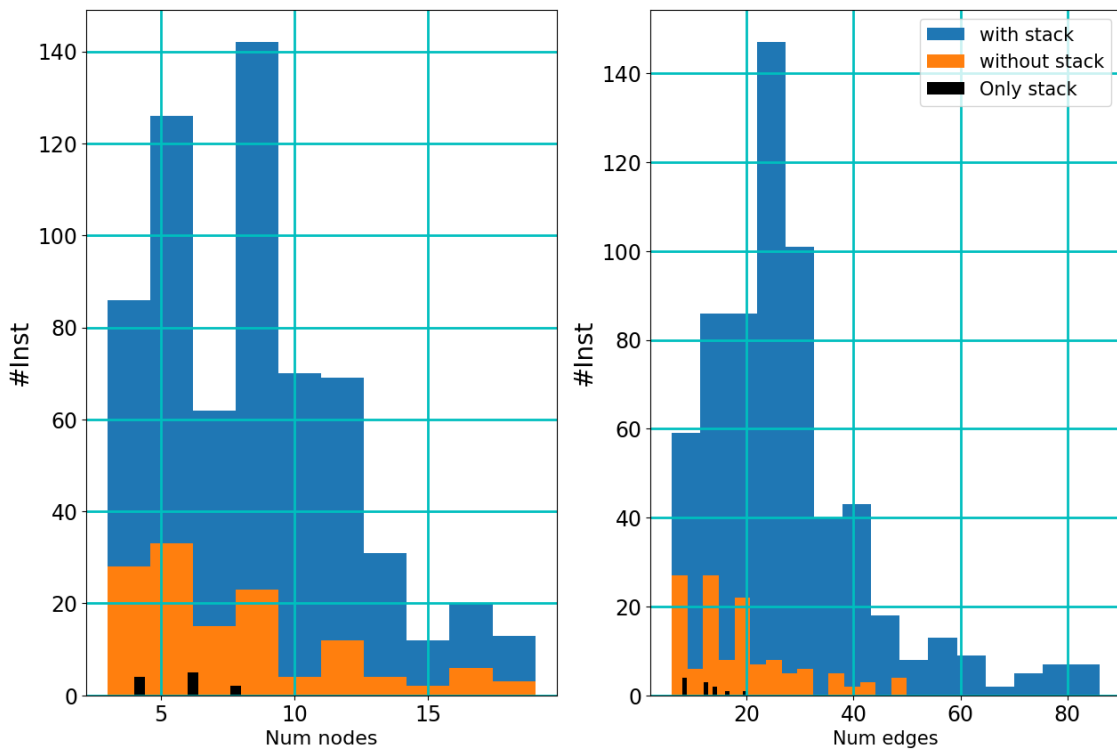


Figure 3.5 Distribution du nombre de nœuds et d'arêtes pour les RINs locaux.

La figure 3.5 présente la distribution du nombre de nœuds et d'arêtes pour les RINs avec et sans empilements. Pour obtenir les RINs sans empilements, j'ai simplement supprimé les empilements des 631 RINs. Nous observons que la diversité des RINs est réduite lorsque les empilements sont exclus par rapport à lorsqu'ils sont inclus. Nous passons de 631 RINs à 130 RINs sans empilements. Nous pouvons également observer qu'il existe des RINs composés uniquement d'empilements; il y a 11 de ces RINs. Le plus grand RIN sans empilements compte 19 nœuds et 50 arêtes, tandis que le plus grand RIN contenant uniquement des empilements compte 8 nœuds et 20 arêtes.

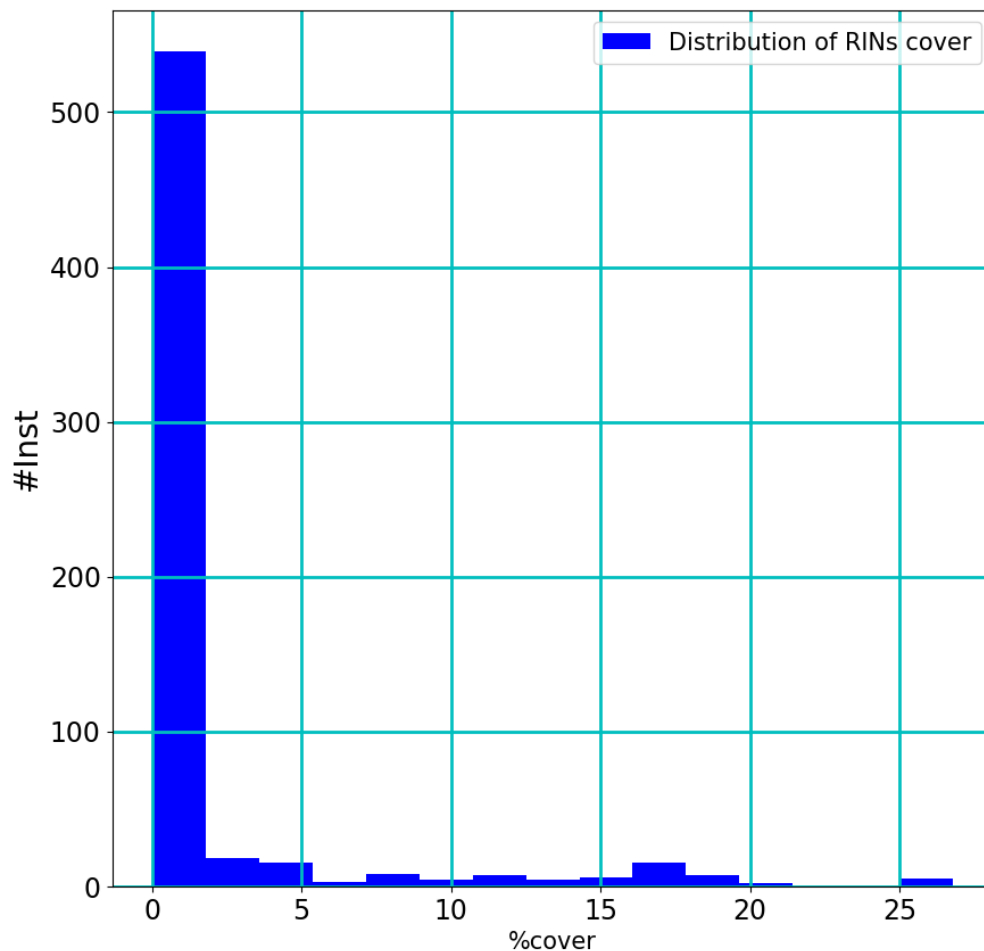


Figure 3.6 Distribution de la couverture des RINs locaux.

La figure 3.6 illustre la distribution de la couverture pour les RINs. La couverture d'un RIN est définie comme le nombre de graphes dans le jeu de données d'origine dans lesquels il apparaît au moins une fois.

3.2.2 RINs Globaux

Le jeu de données global se compose de 1 722 structures d'ARN représentant des structures entières. J'ai trouvé 157 344 RINs pour un total de 209 750 474 occurrences. Le RIN le plus petit a 3 nœuds et 6 arêtes, et le plus grand en a 1 135 nœuds et 5 566 arêtes.

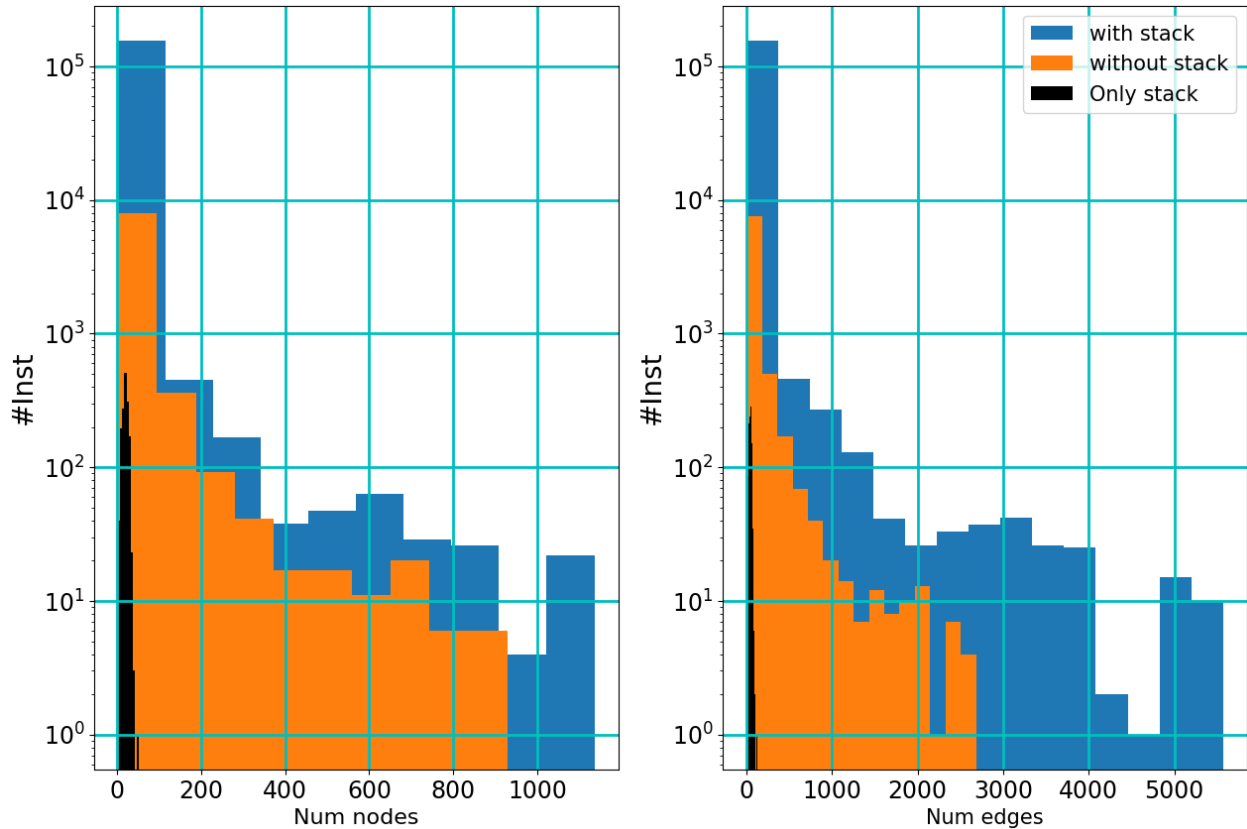


Figure 3.7 Distribution du nombre de nœuds et d'arêtes pour les RINs globaux.

Les figures 3.7 et 3.8 illustrent respectivement la distribution du nombre de nœuds, d'arêtes et la couverture des RINs. De manière similaire aux RINs locaux, une réduction de la diversité des RINs sans empilements est observée, passant de 157 344 RINs à 8 484 RINs. J'ai également 1 525 RINs contenant uniquement des empilements. Le plus grand RIN sans empilements compte 929 nœuds et 2 684 arêtes, tandis que le plus grand RIN contenant uniquement des empilements compte 53 nœuds et 126 arêtes.

La figure 3.9 montre les variations du motif A-MINOR avec empilements ayant été retrouvé dans les RINs globaux. J'ai trouvé 15 motifs différents, tous inclus dans seulement 2 motifs. Les chiffres sur la figure indiquent la couverture des motifs.

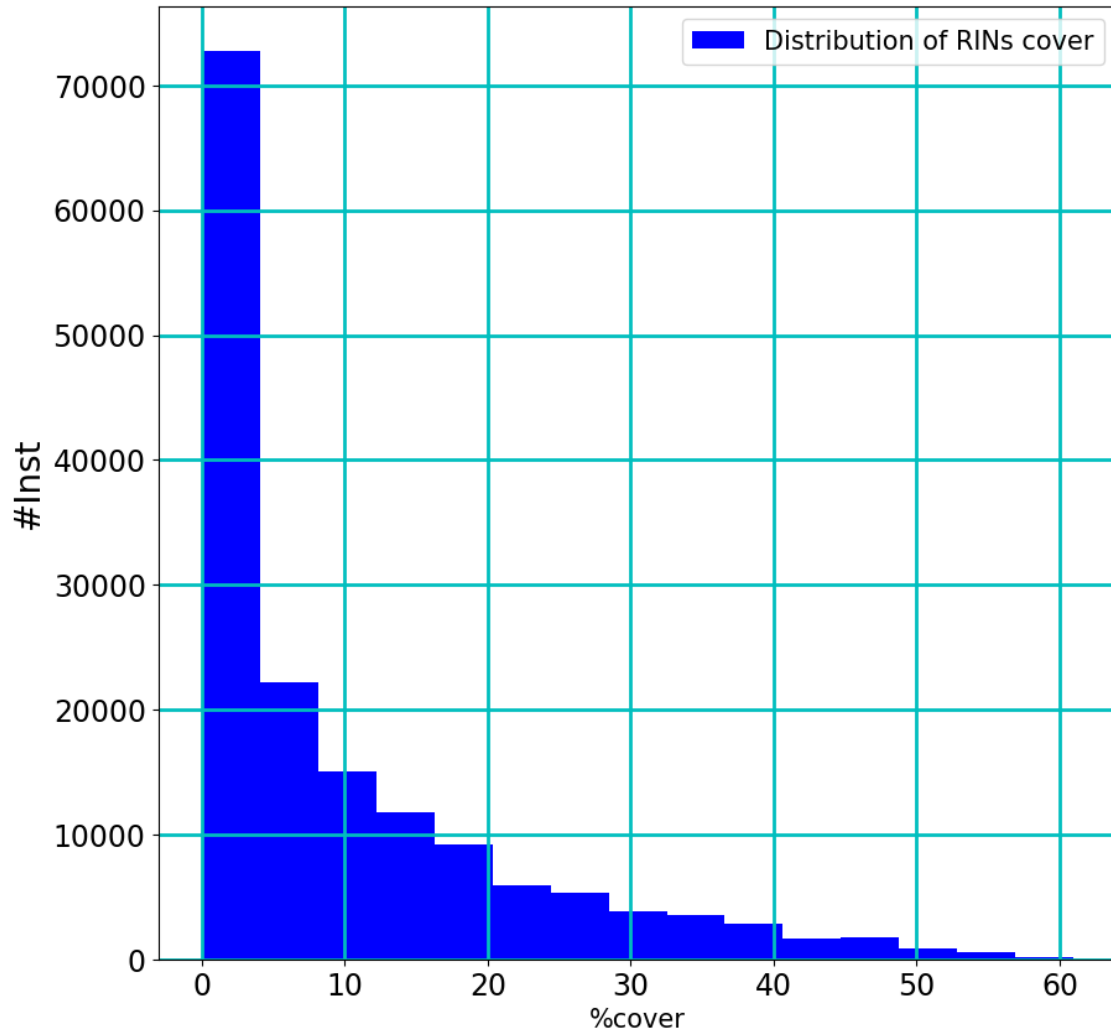


Figure 3.8 Répartition de la couverture des RINs globaux.

J'ai utilisé l'extension d'un motif pour effectuer une comparaison par paire des structures dans l'ensemble de données global afin d'identifier les RINs contenant le motif RIN1 (Figure 3.10). Cela a permis de découvrir 5 variations du motif Kink-turn U4 snRNA (Huang et Lilley, 2016) avec des empilements. J'ai également trouvé 21 variations du motif Kink-turn Hm Kt-7 (Huang et Lilley, 2016) avec des empilements. Ces motifs Kink-turn ne sont pas présents dans les RINs globaux car les contraintes sur les RINs utilisées pour restreindre l'espace de recherche empêchent leur identification (dans PASgraph, ces contraintes peuvent être désactivées ou relâchées pour faire une recherche plus complète).

L'extension d'un motif est intéressante car elle permet de cibler des RINs spécifiques en trouvant des RINs contenant un motif donné. Le ciblage de RINs spécifiques peut réduire considérablement l'espace de re-

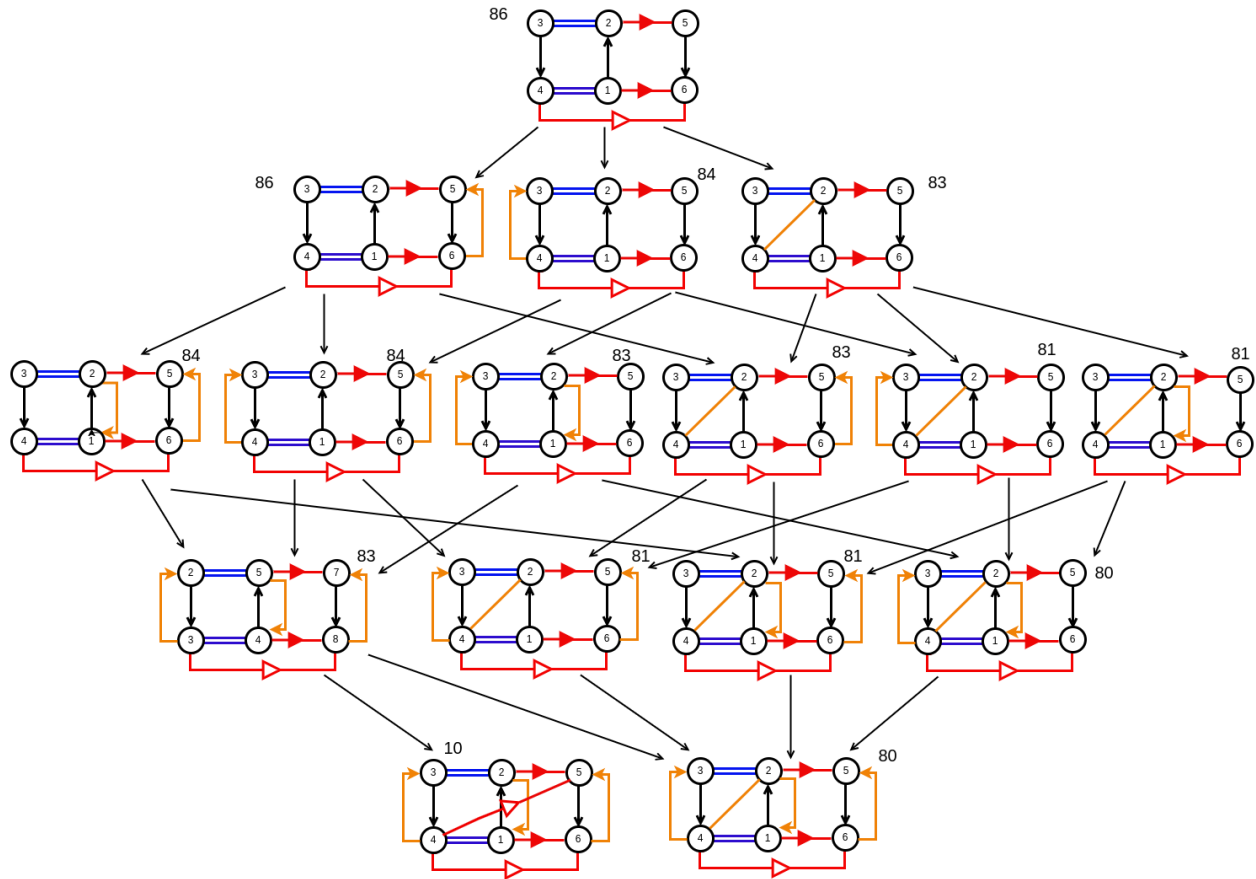


Figure 3.9 Variation des motifs A-MINOR avec empilement. Les lignes orange orientées et non orientées représentent respectivement des interactions d'empilement S53 et S55.

cherche et donc être plus rapide que la recherche de RINs sans aucune connaissance préalable.

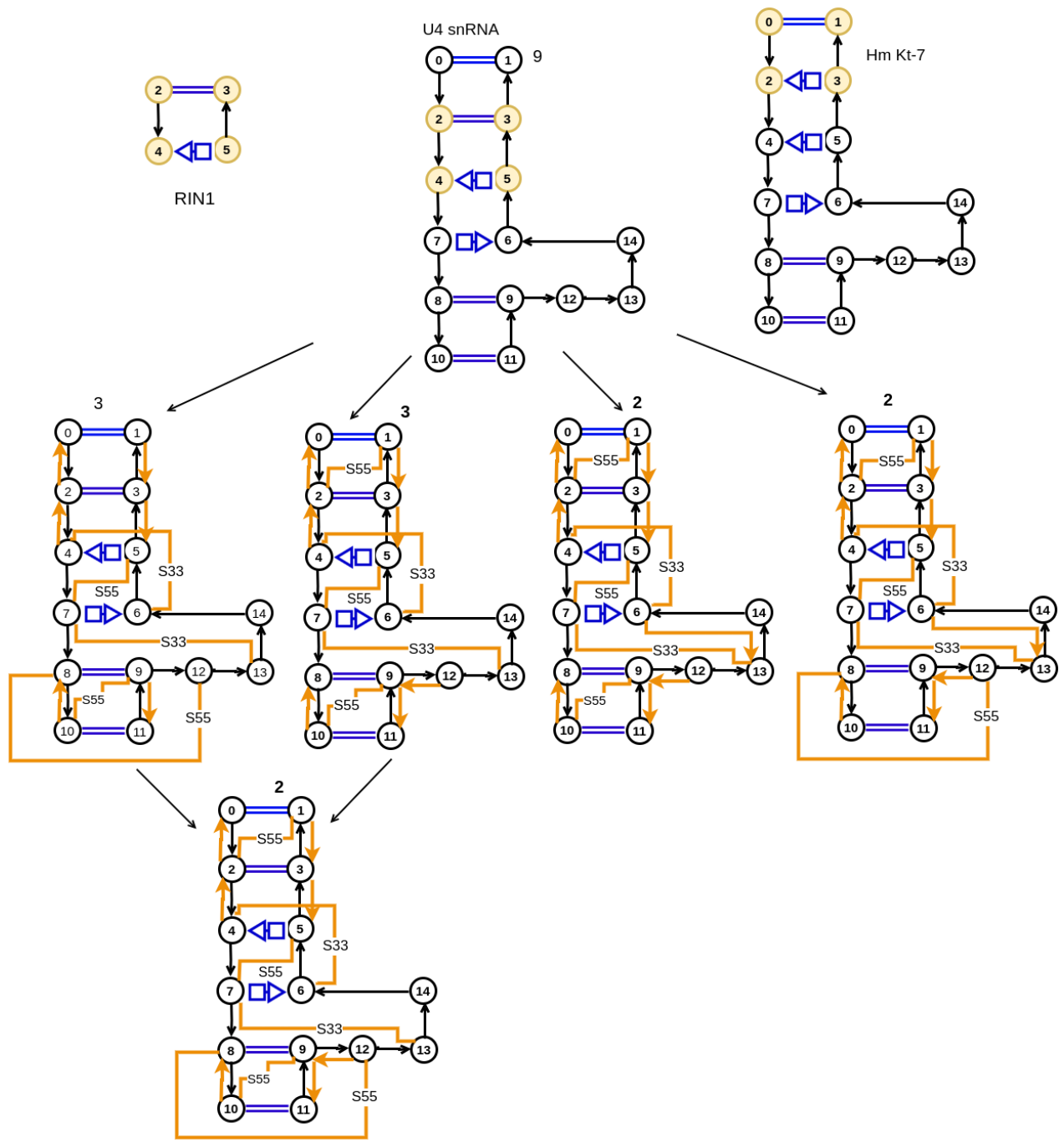


Figure 3.10 Variation des motifs Kink-turn U4 snRNA avec empilement
 En étendant le motif RIN1, nous avons découvert les motifs Kink-turn U4 snRNA et Hm Kt-7.

3.3 Comparaison avec les travaux précédents

Bien que l'ensemble de données que j'ai utilisé provient d'une version différente de RNA3DHub que celle utilisée par les projets (Reinharz *et al.*, 2018; Soulé *et al.*, 2021), je valide mes résultats par rapport aux leurs. Malgré la différence de version, ces ensembles de données devraient être similaires.

3.3.1 Anciens Rins locaux

Les anciens RINs locaux sont au nombre de 389, avec seulement 115 d'entre eux présents dans nos 633 RINs locaux (j'ai également 115 présents dans l'ensemble de données local, validant ainsi ces résultats). Cependant, ils sont tous présents dans les 157 344 RINs globaux. Donc on peut conclure que, les anciens RINs locaux ne sont pas tous présents dans nos RINs locaux en raison du découpage (l'ensemble de données local est obtenu en découpant les structures entières d'ARN de l'ensemble de données global en boucles multiples, boucles intérieures, renflements et en épingles à cheveux).

3.3.2 Anciens Rins globaux

Le nombre total d'anciens RINs globaux est de 1875, parmi lesquels seuls 32 ne sont pas présents dans les RINs globaux. Parmi ces 32 RINs, 6 ne sont présents dans aucun ARN de l'ensemble de données global, et 26 sont présents en une seule occurrence dans un seul ARN de l'ensemble de données global. Cela explique pourquoi ces 32 anciens RINs globaux ne sont pas retrouvés dans les RINs globaux.

CONCLUSION

Dans ce document, je présente une méthode automatisée et distribuée pour extraire des motifs structuraux récurrents d'ARN basée sur leurs interactions plutôt que sur leur séquence ou leur contexte. En utilisant une représentation graphique de l'ARN, je définis les RINs comme des MCCPS (Sous-graphes Partiels Communs Connexes Maximaux) entre deux structures d'ARN. Je conçois un algorithme pour identifier les MCCPS entre deux graphes représentant des structures d'ARN. Grâce à la programmation parallèle, ma méthode améliore significativement le temps de calcul par rapport aux approches précédentes, me permettant, pour la première fois, d'incorporer les interactions d'empilement dans la recherche des RINs.

J'ai appliqué ma méthode à toutes les paires de structures d'ARN non redondantes pour identifier tous les RINs dans deux contextes distincts : (1) motifs de boucles locaux en ne comparant que des boucles multiples, des boucles intérieures, des renflements et des épingles à cheveux, et (2) motifs globaux où un ensemble de structures entières d'ARN ont été comparé entre elles. J'ai trouvé 633 motifs locaux et 157 344 motifs globaux. Ces résultats n'étaient pas réalisables avec les Méthodes proposées (ma méthode est beaucoup plus rapide) dans les travaux précédents (Reinharz *et al.*, 2018; Soulé *et al.*, 2021).

Mon approche permet de démontrer l'importance des interactions d'empilement. En effet, la diversité des motifs diminue significativement lorsque l'on supprime les interactions d'empilement. Le nombre de motifs locaux passe de 633 à 130, et celui des motifs globaux passe de 157 344 à 8 484. Je trouve même des motifs entièrement composés que d'interactions d'empilement. De plus, je mets en valeur la diversité de motifs connus tels que le A-minor et le kink-turn avec des interactions d'empilement.

Mon approche ouvre la voie à une base de données de motifs d'ARN, englobant des motifs locaux et des motifs d'interactions, qui peut être mise à jour fréquemment et rapidement. Cela contribuera également au développement d'outils robustes pour la modélisation et la prédiction de la structure d'ARN.

Les orientations de futures recherches pour cette étude comprennent : La conception d'une interface conviviale pour visualiser, explorer et interagir avec les motifs d'ARN identifiés. Une interface utilisateur intuitive faciliterait la compréhension des résultats et permettrait aux utilisateurs d'explorer les motifs d'ARN de manière plus approfondie. Il est également possible d'élargir la recherche pour inclure des motifs basés sur d'autres types d'interactions présentes dans les ARN afin de capturer plus de diversité dans les motifs d'ARN.

En explorant différents types d'interactions, nous pourrions découvrir de nouveaux motifs d'ARN significatifs qui n'ont pas été identifiés précédemment. Il serait aussi intéressant de continuer à améliorer les performances de mon algorithme pour réduire encore le temps de calcul, notamment pour les structures d'ARN grandes et denses. Une approche prometteuse serait de développer d'autres stratégies d'ordonnancement des tâches ainsi que d'autres techniques de communication comme la communication RDMA (Remote Direct Memory Access). Mon approche peut également être appliquée à d'autres ensembles de réseaux biologiques pour identifier des motifs dans des contextes biologiques spécifiques, tels que la régulation génique ou les voies biologiques.

ANNEXE A
ANNEXE

Les figures ci-dessous représentent la distribution du nombre d'arêtes pour les graphes de l'ensemble de données global et l'ensemble de données local.

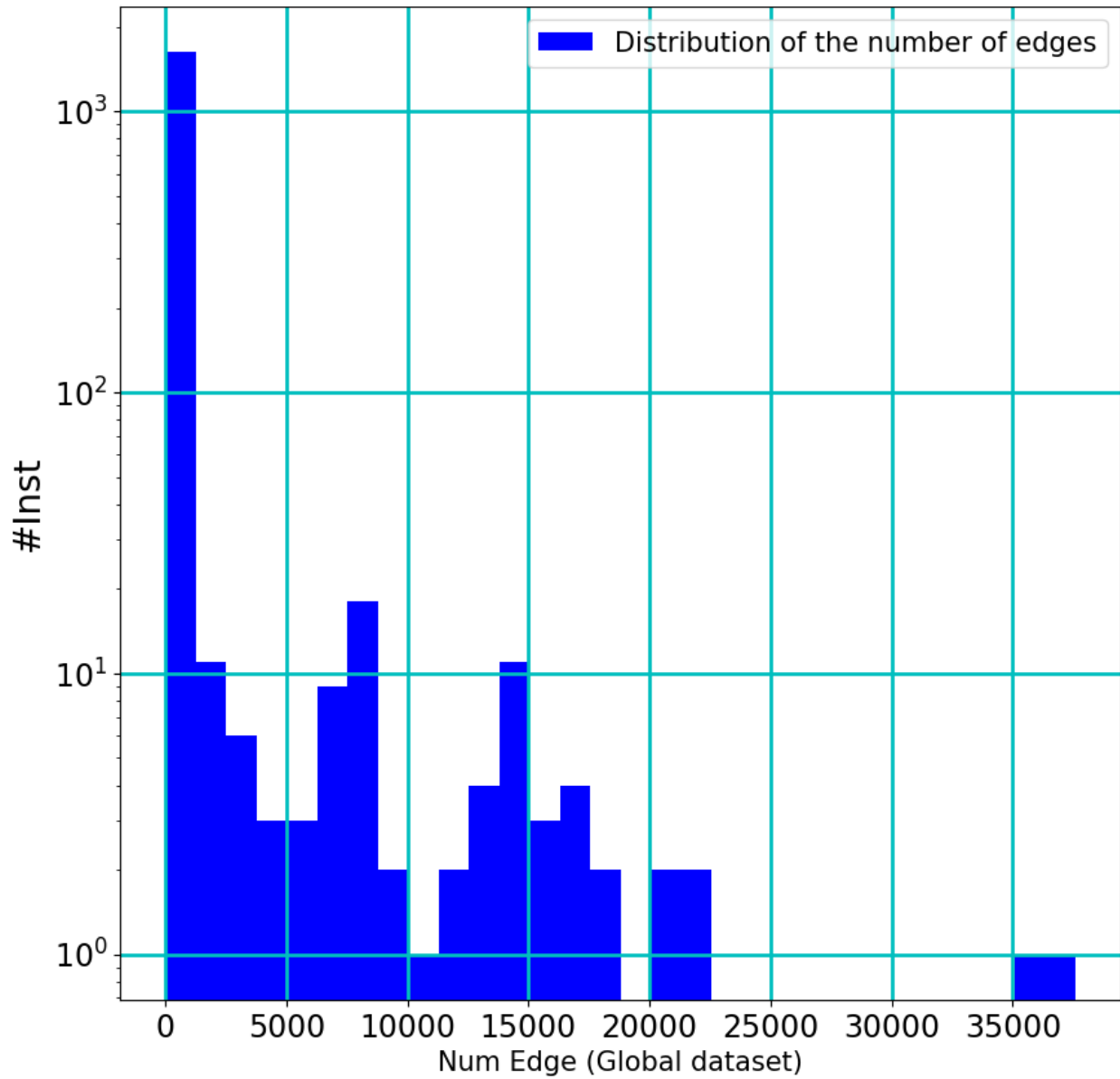


Figure A.1 Distribution du nombre d'arêtes pour l'ensemble de données global à l'échelle logarithmique.

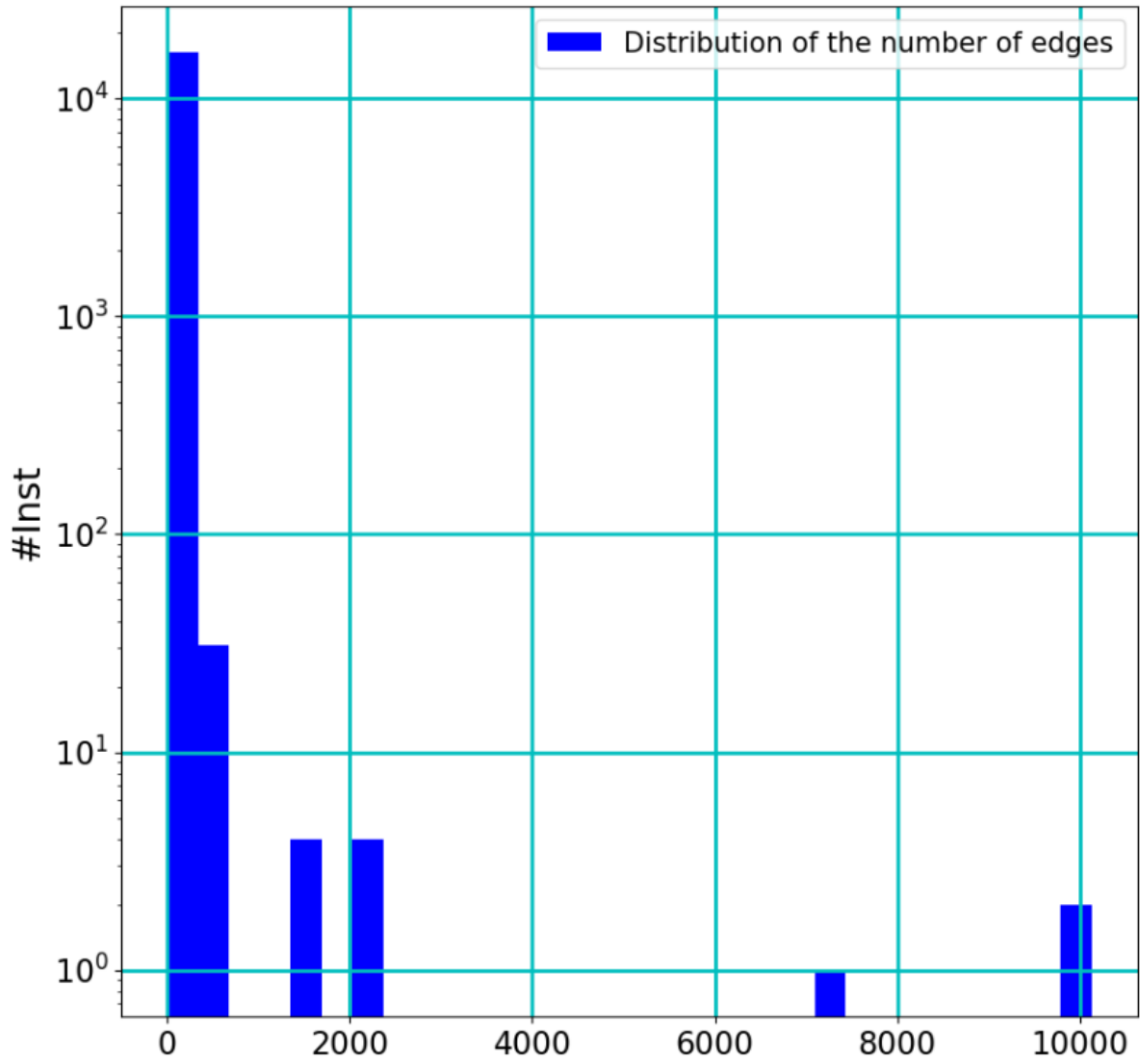


Figure A.2 Distribution du nombre d'arêtes pour l'ensemble de données local à l'échelle logarithmique.

BIBLIOGRAPHIE

- André, É., Coti, C. et Nguyen, H. G. (2015a). Enhanced distributed behavioral cartography of parametric timed automata. Dans *Proceedings of The 17th International Conference on Formal Engineering Methods (ICFEM 2015)*, 319–335. http://dx.doi.org/10.1007/978-3-319-25423-4_21. Récupéré de http://dx.doi.org/10.1007/978-3-319-25423-4_21
- André, É., Coti, C. et Nguyen, H. G. (2015b). Enhanced Distributed Behavioral Cartography of Parametric Timed Automata (Informal Presentation). Dans É. André et G. Frehse (dir.). *2nd International Workshop on Synthesis of Complex Parameters (SynCoP'15)*, volume 44 de *OpenAccess Series in Informatics (OASICS)*, 104–105., Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. <http://dx.doi.org/http://dx.doi.org/10.4230/OASICS.SynCoP.2015.104>. Récupéré de <http://drops.dagstuhl.de/opus/volltexte/2015/5613>
- Bahiense, L., Manić, G., Piva, B. et De Souza, C. C. (2012). The maximum common edge subgraph problem : A polyhedral investigation. *Discrete Appl. Math.*, 160(18), 2523–2541. <http://dx.doi.org/10.1016/j.dam.2012.01.026>. Récupéré de <https://doi.org/10.1016/j.dam.2012.01.026>
- Coimbatore, B., Westbrook, J., Ghosh, S., Petrov, A. I., Sweeney, B., Zirbel, C. L., Leontis, N. B. et Berman, H. M. (2013). The Nucleic Acid Database : new features and capabilities. *Nucleic Acids Research*, 42(D1), D114–D122. <http://dx.doi.org/10.1093/nar/gkt980>. Récupéré de <https://doi.org/10.1093/nar/gkt980>
- Conte, D., Foggia, P. et Vento, M. (2007). Challenging complexity of maximum common subgraph detection algorithms : A performance analysis of three algorithms on a wide database of graphs. *J. Graph Algorithms Appl.*, 11, 99–143. <http://dx.doi.org/10.7155/jgaa.00139>
- Cook, S. A. (1971). The complexity of theorem-proving procedures, stoc'71 : Proceedings of the third annual acm symposium on theory of computing.
- Coti, C., Monniaux, D. et Yu, H. (2019). Parallel parametric linear programming solving, and application to polyhedral computations. Dans J. M. F. Rodrigues, P. J. S. Cardoso, J. Monteiro, R. Lam, V. V. Krzhizhanovskaya, M. H. Lees, J. J. Dongarra, et P. M. Sloot (dir.). *Computational Science – ICCS 2019*, 566–572., Cham. Springer International Publishing.
- Darty, K., Denise, A. et Ponty, Y. (2009). Varna : Interactive drawing and editing of the rna secondary structure. *Bioinformatics (Oxford, England)*, 25(15), 1974–1975. <http://dx.doi.org/10.1093/bioinformatics/btp250>. Récupéré de <https://doi.org/10.1093/bioinformatics/btp250>
- Delano, W. (2006). Pymol home page, pymol. sourceforge. net.
- Depolli, M., Konc, J., Rozman, K., Trobec, R. et Janezic, D. (2013). Exact parallel maximum clique algorithm for general and protein graphs. *Journal of chemical information and modeling*, 53. <http://dx.doi.org/10.1021/ci4002525>
- Djelloul, M. (2009). *Algorithmes de graphes pour la recherche de motifs récurrents dans les structures*

tertiaires d'ARN. (Theses). Université Paris Sud - Paris XI. Récupéré de <https://tel.archives-ouvertes.fr/tel-00785953>

- Gianfrotta, C. (2022). *Modélisation, analyse et classification de motifs structuraux d'ARN à partir de leur contexte, par des méthodes d'algorithmique de graphes*. (Bio-informatique [q-bio.qm]). Université Paris-Saclay. NNT : 2022UPASG056.
- Glouzon, J.-P. S. (2017). *Exploration des structures secondaires de l'ARN*. (Theses). Université De Sherbrooke. Récupéré de <http://hdl.handle.net/11143/10672>
- Hendrix, D. K., Brenner, S. E. et Holbrook, S. R. (2005). Rna structural motifs : building blocks of a modular biomolecule. *Q Rev Biophys*, 38(3), 221-243.
<http://dx.doi.org/10.1017/S0033583506004215>
- Hermann, T. et Patel, D. J. (2000). Rna bulges as architectural and recognition motifs. *Structure*, 8(3), R47-R54. [http://dx.doi.org/https://doi.org/10.1016/S0969-2126\(00\)00110-6](http://dx.doi.org/https://doi.org/10.1016/S0969-2126(00)00110-6).
Récupéré de <https://www.sciencedirect.com/science/article/pii/S0969212600001106>
- Holbrook, S. R. (2005). Rna structure : the long and the short of it. *Curr Opin Struct Biol*, 15(3), 302-308.
<http://dx.doi.org/10.1016/j.sbi.2005.04.005>
- Huang, L. et Lilley, D. M. (2016). The kink turn, a key architectural element in rna structure. *Journal of Molecular Biology*, 428(5, Part A), 790-801. Challenges in RNA Structural Modeling and Design, <http://dx.doi.org/https://doi.org/10.1016/j.jmb.2015.09.026>. Récupéré de <https://www.sciencedirect.com/science/article/pii/S0022283615005495>
- Huang, X., Lai, J. et Jennings, S. (2006). Maximum common subgraph : Some upper bound and lower bound results. *BMC bioinformatics*, 7 Suppl 4, S6.
<http://dx.doi.org/10.1186/1471-2105-7-S4-S6>
- Kann, V. (1992). On the approximability of the maximum common subgraph problem. 377-388.
http://dx.doi.org/10.1007/3-540-55210-3_198
- Leontis, N., Stombaugh, J. et Westhof, E. (2002a). Motif prediction in ribosomal rnas lessons and prospects for automated motif prediction in homologous rna molecules. *Biochimie*, 84(9), 961-973.
[http://dx.doi.org/https://doi.org/10.1016/S0300-9084\(02\)01463-3](http://dx.doi.org/https://doi.org/10.1016/S0300-9084(02)01463-3). Récupéré de <https://www.sciencedirect.com/science/article/pii/S0300908402014633>
- Leontis, N., Stombaugh, J. et Westhof, E. (2002b). The non-watson-crick base pairs and their associated isostericity matrices. *Nucleic acids research*, 30, 3497-531.
<http://dx.doi.org/10.1093/nar/gkf481>
- Leontis, N. B., Lescoute, A. et Westhof, E. (2006). The building blocks and motifs of rna architecture. *Current Opinion in Structural Biology*, 16(3), 279-287. Nucleic acids/Sequences and topology, <http://dx.doi.org/https://doi.org/10.1016/j.sbi.2006.05.009>. Récupéré de <https://www.sciencedirect.com/science/article/pii/S0959440X06000807>
- Leontis, N. B., Stombaugh, J. et Westhof, E. (2002c). The non-Watson-Crick base pairs and their

- associated isostericity matrices. *Nucleic Acids Research*, 30(16), 3497–3531.
<http://dx.doi.org/10.1093/nar/gkf481>. Récupéré de
<https://doi.org/10.1093/nar/gkf481>
- Leontis, N. B. et Westhof, E. (2002). The annotation of rna motifs. *Comparative and functional genomics*, 3(6), 518–524. <http://dx.doi.org/10.1002/cfg.213>
- Leontis, N. B. et Westhof, E. (2003). Analysis of rna motifs. *Current Opinion in Structural Biology*, 13(3), 300–308. [http://dx.doi.org/https://doi.org/10.1016/S0959-440X\(03\)00076-9](http://dx.doi.org/https://doi.org/10.1016/S0959-440X(03)00076-9).
Récupéré de
<https://www.sciencedirect.com/science/article/pii/S0959440X03000769>
- Lescoute, A., Leontis, N. B., Massire, C. et Westhof, E. (2005a). Recurrent structural RNA motifs, Isostericity Matrices and sequence alignments. *Nucleic Acids Research*, 33(8), 2395–2409.
<http://dx.doi.org/10.1093/nar/gki535>. Récupéré de
<https://doi.org/10.1093/nar/gki535>
- Lescoute, A., Leontis, N. B., Massire, C. et Westhof, E. (2005b). Recurrent structural RNA motifs, Isostericity Matrices and sequence alignments. *Nucleic Acids Research*, 33(8), 2395–2409.
<http://dx.doi.org/10.1093/nar/gki535>. Récupéré de
<https://doi.org/10.1093/nar/gki535>
- McCreesh, C. et Prosser, P. (2013). Multi-threading a state-of-the-art maximum clique algorithm. *Algorithms*, 6, 618–635. <http://dx.doi.org/10.3390/a6040618>
- Minot, M., Ndiaye, S. N. et Solnon, C. (2015). Recherche d'un plus grand sous-graphe commun par décomposition du graphe de compatibilité. Dans *Onzièmes Journées Francophones de Programmation par Contraintes (JFPC)*, 1–11., Bordeaux, France. Récupéré de
<https://hal.science/hal-01163322>
- Ndiaye, S. N. et Solnon, C. (2011). Cp models for maximum common subgraph problems. Dans *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming, CP'11*, p. 637–644., Berlin, Heidelberg. Springer-Verlag.
- Nissen, P., Ippolito, J. A., Ban, N., Moore, P. B. et Steitz, T. A. (2001). Rna tertiary interactions in the large ribosomal subunit : The a-minor motif. *Proceedings of the National Academy of Sciences*, 98(9), 4899–4903. <http://dx.doi.org/10.1073/pnas.081082398>. Récupéré de
<https://www.pnas.org/doi/abs/10.1073/pnas.081082398>
- Petrov, A., Zirbel, C. et Leontis, N. (2013). Automated classification of rna 3d motifs and the rna 3d motif atlas. *RNA (New York, N.Y.)*, 19. <http://dx.doi.org/10.1261/rna.039438.113>
- Quer, S., Marcelli, A. et Squillero, G. (2020). The maximum common subgraph problem : A parallel and multi-engine approach. *Computation*, 8, 48.
<http://dx.doi.org/10.3390/computation8020048>
- Raymond, J., Gardiner, E. et Willett, P. (2002). Rascal : Calculation of graph similarity using maximum common edge subgraphs. *Comput. J.*, 45, 631–644.

- Raymond, J. et Willett, P. (2002). Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of computer-aided molecular design*, 16, 521–33.
<http://dx.doi.org/10.1023/A:1021271615909>
- Reinharz, V., Soulé, A., Westhof, E., Waldispühl, J. et Denise, A. (2018). Mining for recurrent long-range interactions in RNA structures reveals embedded hierarchies in network families. *Nucleic Acids Research*, 46(8), 3841–3851. <http://dx.doi.org/10.1093/nar/gky197>. Récupéré de <https://doi.org/10.1093/nar/gky197>
- Sarver, M., Zirbel, C. L., Stombaugh, J., Mokdad, A. et Leontis, N. B. (2007). Fr3d : finding local and composite recurrent structural motifs in rna 3d structures. *Journal of Mathematical Biology*, 56, 215–252. Récupéré de <https://api.semanticscholar.org/CorpusID:31595908>
- Sarver, M., Zirbel, C. L., Stombaugh, J., Mokdad, A. et Leontis, N. B. (2008). Fr3d : finding local and composite recurrent structural motifs in rna 3d structures. *Journal of mathematical biology*, 56(1-2), 215–252. <http://dx.doi.org/10.1007/s00285-007-0110-x>
- Schmidt, R., Klein, R. et Rarey, M. (2022). Maximum common substructure searching in combinatorial make-on-demand compound spaces. *Journal of Chemical Information and Modeling*, 62(9), 2133–2150. PMID : 34478299, <http://dx.doi.org/10.1021/acs.jcim.1c00640>. Récupéré de <https://doi.org/10.1021/acs.jcim.1c00640>
- Shalybkova, A. A., Mikhailova, D. S., Kulakovskiy, I. V., Fakhranurova, L. I. et Baulin, E. F. (2021). Annotation of the local context of the rna secondary structure improves the classification and prediction of a-minors. *RNA (New York, N.Y.)*, 27(8), 907–919. Advance online publication, <http://dx.doi.org/10.1261/rna.078535.120>
- Soulé, A., Reinharz, V., Sarrazin-Gendron, R., Denise, A. et Waldispühl, J. (2021). Finding recurrent RNA structural networks with fast maximal common subgraphs of edge-colored graphs. *PLOS Computational Biology*, 17(5), 1–28. <http://dx.doi.org/10.1371/journal.pcbi.1008990>. Récupéré de <https://doi.org/10.1371/journal.pcbi.1008990>
- Wadley, L. M. et Pyle, A. M. (2004). The identification of novel RNA structural motifs using COMPADRES : an automated approach to structural discovery. *Nucleic Acids Research*, 32(22), 6650–6659. <http://dx.doi.org/10.1093/nar/gkh1002>. Récupéré de <https://doi.org/10.1093/nar/gkh1002>
- Wang, X., Huan, J., Snoeyink, J. S. et Wang, W. (2007). Mining rna tertiary motifs with structure graphs. Dans *19th International Conference on Scientific and Statistical Database Management (SSDBM 2007)*, 31–31. <http://dx.doi.org/10.1109/SSDBM.2007.38>
- Zhong, C. et Zhang, S. (2011). Clustering RNA structural motifs in ribosomal RNAs using secondary structural alignment. *Nucleic Acids Research*, 40(3), 1307–1317. <http://dx.doi.org/10.1093/nar/gkr804>. Récupéré de <https://doi.org/10.1093/nar/gkr804>
- Zirbel, C. L., Sponer, J. E., Sponer, J., Stombaugh, J. et Leontis, N. B. (2009). Classification and energetics of the base-phosphate interactions in rna. *Nucleic acids research*, 37(15), 4898–4918. <http://dx.doi.org/10.1093/nar/gkp468>